Date of acceptance Grade

Instructor

Sentence segmentation on poorly structured text using language models

Eeva Nikkari

Helsinki April 24, 2017

MSc Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Eeva Nikkari

Työn nimi — Arbetets titel — Title

Sentence segmentation on poorly structured text using language models

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc Thesis	April 24, 2017	49 pages + 10 appendices

Tiivistelmä — Referat — Abstract

The sentence segmentation task is the task of segmenting a text corpus into sentences. Segmenting well structured and fully punctuated data into sentences is not a very difficult problem. However, when the data is poorly structured or missing punctuation the task is more difficult. This thesis will look into this problem by using probabilistic language modeling, with special emphasis on the n-gram model. We will present theory related to language models and evaluating them, as well as empirical results achieved on documents provided by AlphaSense Oy and a freely available Reuters-21578 corpus.

The experiments on n-gram models focused on the following questions. How does the smoothing and order of the n-gram affect the model? How well does a model trained on one type of data adapt to another type of text? How does retaining more or less symbols and punctuation affect the performance? And how much is enough training data for the model?

The n-gram models performed rather well on the same type of data they were trained on. However, the performance was significantly worse when moving to another document type. In absence of punctuation the performance of the model was also rather poor. The conclusion is that the n-gram model seems inadequate in recovering the sentence boundaries in difficult settings such as separating the unpuncutated title from the body of the text.

ACM Computing Classification System (CCS):
Computing methodologies - Artificial intelligence - Natural language processing
Mathematics of computing - Probability and statistics - Stochastic processes - Markov processes

Avainsanat — Nyckelord — Keywords
sentence boundary disambiguation, SBD, sentence segmentation, Machine Learning, Language Modeling

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Natural language processing (NLP) is a field of machine learning concerned with processing human natural languages such as English or Finnish. The data used in NLP is usually either from speech recognition systems or text corpora such as newspaper stories, novels or comments on social media. Well known NLP problems are for example sentiment analysis, in which the system tries to understand the sentiment of text or spoken commands; and machine translation in which the system tries to translate text from one language to another.

So far humans outperform NLP systems in understanding natural languages: most us of us have tried NLP systems, such as Google translate or Siri, and know them to be imperfect albeit impressive. Despite their flaws, the need for NLP systems arises from the need to process either huge amounts of data, or processing data fast. After all, humans are very slow at processing data compared to a computational system. This makes it impossible to use humans in processing large datasets, or fast query answering. For an overview of the NLP field refer to articles [HM15] and [NC11].

Sentence segmentation is an important first step in many NLP systems. This means finding the places in text where a sentence ends and another begins. A sentence is an element of text that contains a concise piece of information; this makes it valuable to an NLP system. Finding the sentence boundaries in a text might seem like a trivial task for humans, but is not for an automated system. Even humans have trouble deciphering sentence boundaries when information such as punctuation or capitalization is missing, or when the data is otherwise poorly structured.

Many of the challenges of NLP come from the qualities of the corpora. The corpora can be either be fully punctuated text, such as novels or newspaper text, or completely unpunctuated such as transcripts produced by automatic speech recognition. Even when working with a fully punctuated corpus, we come across ambiguities on whether a period denotes a sentence boundary or not: abbreviations that contain periods, sentences that end in an abbreviation and sentences inside quotes. Also the text type presents problems: the vocabulary and sentence structure can differ drastically between text types such as newspaper text, cooking instructions, messages on social media and novels. The documents used for experiments in this thesis are all more or less newspaper type text that are focused on financial themes.

This thesis will take a look into probabilistic language modeling and the sentence segmentation task. Language models and how they can be can be applied to the

sentence segmentation task are discussed in Chapter 2. This thesis will cover ways to evaluate language models and a segmentation in Chapter 3. We will focus in more detail on the n-gram model in Chapter 4. In Chapter 5 we will introduce the different datasets and in Chapter 6 present some empirical results.

This thesis work is done for AlphaSense Oy [1] and the documents used in experiments were provided by AlphaSense, except for the freely available Reuters-21578 dataset. AlphaSense is a Financial Search Engine that contains documents from several different providers. The datasets are presented and compared to each other in Chapter 5.3. The experiments in Chapter 6 are motivated by the interest in segmenting a text type referred to as D that can't be segmented properly by the same methods as the other documents.

The AlphaSense documents are not originally pure text documents, which causes problems in their structure: the documents contain ambiguous structures such as address-lines, data tables and titles that don't end in periods. We will present the different document types in Chapter 5.3. Even though the documents all represent similar text types, there was a drastic decrease in performance when using a language model trained on one type of data on another type of data in the experiments in Chapter 6.3.

AlphaSense has a continuous stream of new documents to the system. For this reason the lightweight n-gram model was chosen to experiment on. More complex models might achieve better segmentation results, but their performance could be too heavy and slow to perform on each document. In the experiments the n-gram model proved to achieve unsatisfactory results and in Chapter 7 we will present some other models used in NLP. However, further experiments on different models is left out of the scope of this thesis. The experiments in Chapter 6 were conducted using the SRILM toolkit. The Appendixes of this thesis contain further information on SRILM.

2 Probabilistic formulation of the sentence boundary disambiguation task

In this chapter we will explain the concept of a language model and how they can be used to recover sentence boundaries.

[1] `https://www.alpha-sense.com/`

2.1 The task description

We will first consider the classification task of discrete sequential data in a more general sense and then explain how it ties to the sentence boundary disambiguation task and some other similar problems. In the next paragraph, the reader can first consider an observation o_i to be a word, sequence of words o_1^m to be a text document of m words, and a class c_i to be either "last word in sentence" or "not last word in sentence".

Our task can be described as follows. We denote the set of possible observations as O; and the set of classes as C. We are given a sequence of m observations $o_1^m = o_1 \dots o_m$, where $o_i \in O$. Our goal is to decipher the hidden m-sequence of classes $c_1^m = c_1 \dots c_m$, where $c_i \in C$, so that each observation has a corresponding class-label.

In the *sentence boundary disambiguation task*, our observations are single words, or punctuation marks; and we have two hidden classes: sentence boundary (<B>) and no-sentence boundary (no-event). That is, C={<B>, no-event}, where the label <B> indicates that the word is the last word in a sentence.

The set of observations is the set of all words in the training data, or a pruned subset of it. In practice we might want to prune out words that appear only very few times and replace them with a <unk> token.

Other similar tasks are recovering punctuation and part of speech tagging. For the *punctuation disambiguation task* (adding commas, periods and question marks to an unpunctuated text), our hidden classes would be C_{punct}={<,>, <.>, <?>, no-event}. And in the *part of speech tagging task*, we would consider our hidden classes to be the word-classes of the language (verb, noun, ..).

Note that an observation o_i does not necessarily have to be a one-dimensional observation of a word. Depending on the language model, we can use multidimensional observations that consist of so called features. For example, when defining *sentence boundaries on speech transcripts*, our observation c_i would consist of both the word and prosodic features of the record. Or, on text we can have additional textual clues such as the capitalization of words, or part of speech tags of words. It depends on the probabilistic model if and how well it can handle multidimensional observations or classes.

In the experiments in this thesis, we take the capitalization information into account by adding tags in front of the words. The capitalization-tags are also considered as

words by the model, since the n-gram model does not handle multidimensional observations. This is a simple and lightweight way to introduce the capitalization information to the n-gram model. On the downside, an n-gram will contain less actual words, as the capitalization tags take up some spots that would otherwise contain an actual word. The tagging and other prepossessing steps are explained further in Chapter 5.2. The n-gram model is presented in Chapter 4.

2.2 Probabilistic language model

A language model M is a joint probability distribution that models the relationship between the observation and class variables and probability of different sequences. A joint probability mass function $p^M(o_1^m, c_1^m)$ is enough to define the distribution. That is, a language model assigns probabilities p^M to any sequence q of observation-classes pairs $q \in \{O \times C\}^{1,..,\infty}$

$$p^M : \{O \times C\}^{1,..,\infty} \rightarrow [0, 1]$$

The language model is usually learned in a supervised fashion from data that already has all observations and class-labels in place. If we assume a modeling approach, that is, we decide on dependencies and independences between the variables that we believe the process that generates the data satisfies, we can use training data to infer the model parameters.

Let θ be the model parameters, x the training data and $L : (\theta, x) \rightarrow [0, 1]$ the likelihood function of the distribution we have assumed. The likelihood function of model p^M is a function of the parameters θ, which is equal to the probability given by the model, under the parameters θ: $L(\theta|x) = p^M(x|\theta)$

The maximum likelihood (ML) approach takes the training corpus and outputs the model's parameters that maximize the the likelihood of the training corpus x. The maximum likelihood parameters are denoted as

$$\hat{\theta}_{ML} = \arg\max_{\theta} L(\theta|x).$$

Finding the maximum likelihood parameters for the n-gram model is derived in Chapter 4.1.

2.3 Assigning the hidden classes using a language model

When the model and parameters of the language model M have been defined, we can predict the hidden sequence of classes of a new observation sequence $o_1^m = o_1 \dots o_m$ by finding the class-sequence $c_1^m = c_1 \dots c_m$ that maximizes the joint probability of the observations and classes

$$\hat{c}_1^m = \arg\max_{c_1^m} p^M(o_1^m, c_1^m).$$

The method used for finding the maximum probability sequence varies by the model. In some cases we can use a dynamic programming algorithm to calculate the best sequence, and in some cases we need to approximate the result using methods such as gradient descent.

3 Evaluation measures in language modeling

Now that we have presented the problem at hand, we need a way to measure the results of different modeling approaches. The model can be tested on a testing set that has not been used in training of the model. There are two quantitative ways to measure the performance of a language model on a testing set. One way is to measure the results achieved on the segmentation task, this can be measured by precision, recall and F-measure. These are presented in Subsection 3.1. Another way is to measure how well the distribution models the testing set. This can be measured with entropy and perplexity. These are presented in Subsection 3.2.

Other measures to consider are the speed of segmentation, the quality of training data and the specifics of the modeling approach. These are discussed in Subsection 3.3. Even humans don't achieve perfect results on the segmentation task, if punctuation is missing. In Subsection 3.4 we address an experiment done by Stevenson and Gaizauskas on how well humans perform on the segmentation task when punctuation is missing.

3.1 Evaluation methods for the results of the classification problem

Consider the sentence segmentation task as a classification problem: we consider that between any two words there is a possible sentence boundary and try to classify

them to sentence boundaries and non-sentence boundaries. We compare the original, correctly segmented document, to the document segmented using the language model. At each assigned and original sentence boundary we pause and compare the segmentation.

A positive value corresponds to any predicted sentence boundary: *true positive (tp)* if it's on a real sentence boundary and *false positive (fp)* if not. Similarly a negative value is when we predict there is no sentence boundary: *true negative (tn)* if there is no sentence boundary in the original document and *false negative (fn)* if we miss a sentence boundary that was in the original.

The notation $|\cdot|$ denotes count of $\cdot$. The typical measures to measure the performance of a classification task are precision, recall and the F_1-measure [MS99] :

$$\begin{aligned} precision : P &= \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false positive}|} \\ recall : R &= \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false negative}|} \\ F_1 - measure : F_1 &= \frac{2}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{P+R} \end{aligned}$$

Precision measures the percentage of correct boundaries out of all boundaries assigned by the classifier. Recall measures percentage of how many sentence boundaries out of all true boundaries the classifier found.

F_1-measure is a measure that gives an average of precision and recall. More accurately it is the harmonic mean of precision and recall. Note that this is different from what we usually consider an average of two numbers. F-measure is high only if both values are high. Figure 1 demonstrates how F_1-measure is sensitive to both P and R: even if one of P or R is large, but the other one is small, the F_1-measure gives small values.

In some older articles the measures of *Accuracy* and *Error* are used. The formulas for accuracy and error are

$$\begin{aligned} accuracy : A &= \frac{|tp| + |tn|}{|tp| + |tn| + |fp| + |fn|} \\ error : E &= \frac{|fp| + |fn|}{|tp| + |tn| + |fp| + |fn|.} \end{aligned}$$

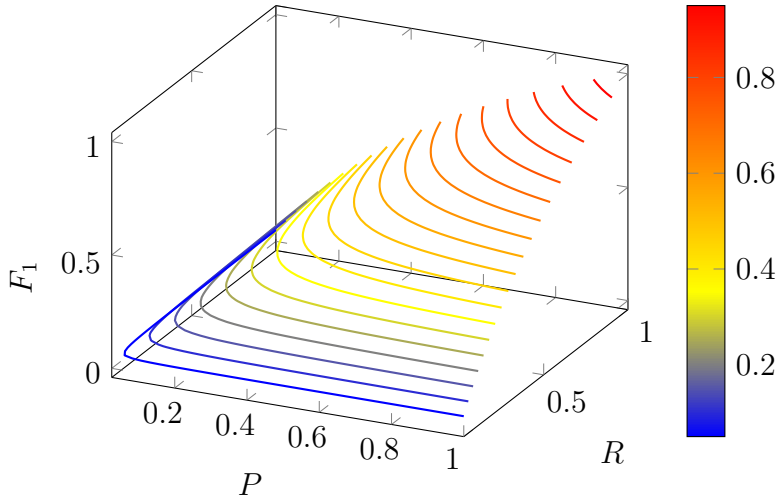

Figure 1: Plot of F_1-measure in regards of P and R;

These two aren't considered to give enough information, since the amount of negatives is overwhelming compared to the amount of positives and you can achieve high accuracy simply by making very few selections. In recent articles the results are measured by the three previous measures.

3.2 Entropy and evaluating language models with perplexity

Not all language modeling tasks can be measured by the classification approach. In many cases it makes more sense to measure how well the model models language. For example, does the model consider typical language to be more likely than untypical? Say, we wanted to implement a NLP system that corrects the word order of a sentence, or corrects spelling. It makes sense to ask if the model considers the unusual or wrong ordering of words, such as "a the roof" to be less likely than a typical one, such as "on the roof". This subsection presents the concept of entropy and how it is used to compare different models of language.

Imagine we are playing a communication game. We have a finite set of events V. We are sequentially given events $w \in V$ and we need to communicate the events forward using as few bits as possible. The events are given to us randomly, according to a distribution $p(w)$. It turns out that the best strategy would be to use as few bits as possible to communicate the most probable events or sequences of events, and more bits on the less likely events. This minimizes the expected amount of bits needed.

The expected amount of bits needed is also called entropy. In this game we unfortu-

nately do not know the true distribution $p(w)$ of the events. We can only get samples from the true distribution. However, if we define artificial distributions $p^{M1}(w)$ and $p^{M2}(w)$ over the same set of events $w \in V$, we can test them on a sufficiently large message $D = w_1 w_2 \dots w_N$ where $w_i \in V$, sampled from the true distribution $p(w)$. Now, the model that can communicate the test message in fewer bits can be considered to be better and closer to the true distribution. This gives us a measure to compare the two models, even when we don't know the true distribution. This is the idea behind comparing entropies of language models.

Next we will define entropy in more formal terms. Let X be a discrete random variable with the probability mass function $p(x)$ in event space S. The entropy of the variable is

$$H(X) = E(\log(\frac{1}{p(X)})) = -\sum_{x \in S} p(x) \log(p(x)).$$

The base of the logarithm is usually 2 when entropy is concerned. It's a weighted average of the uncertainty associated with each event. This can be thought of as a measure of how much information the random variable contains.

Now, instead of the true distribution p of the events, we encode the messages using the distribution $p^M(x)$ over the same event space S. The expected length of the message under the true distribution is called the cross entropy between distributions p and p^M. Formally:

$$H(p, p^M) = E_p\left(\log \frac{1}{p^M(X)}\right) = -\sum_{x \in S} p(x) \log p^M(x).$$

As mentioned before, we do not know the true distribution $p(x)$. However, we can sample a large sequence $D = (X_i)_{i=1}^n = X_1, X_2, \dots, X_n$ from the distribution.

Entropy is the expected number of bits needed to communicate the message $D = (X_i)_{i=1}^n$. Since a large message needs more bits, it makes sense to consider what is the expected message length per event. This is called entropy rate. To define entropy rate we will make some assumptions of the model

We consider language as a discrete *stochastic process*. This means we assume the events of the language form a dependent sequence of indexed events $D = (X_1, X_2, \dots, X_n)$. For example, if we wrote down each word an individual speaks, these would form a stochastic process where the first event is the their first word and all following words are indexed as a new state. The process is characterized by the joint distribution of the events $p(x_1^n) := p(x_1, x_2, \dots, x_n)$, $x_i \in S$. We will present some theory on stochastic processes based on [CT06].

In addition we assume the process is *stationary*, which means that the joint probability of a subsequence of random variables is not dependent on the time index. In the speech example this would mean that the probability of words and their order is not dependent on the time index they are spoken. For an individual this is not true, as language depends significantly on the age and time period. Similarly all natural language depend on the time period and text type. However, when working with natural language processing, we usually limit ourselves to a certain time period and text type, so making this kind of assumption is not too far off. Formally this means that a stationary stochastic process has the following property for all events $x_i \in S$ indexes n and all shifts in time $l \in \mathbb{Z}$:

$$p(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = p(X_{1+l} = x_1, X_{2+l} = x_2, \ldots, X_{n+l} = x_n).$$

We also assume that the process is *ergodic*. This means that the process will not get stuck in a subset of events from which it can never again diverge from. This is a logical assumption for natural language, since there are no words we could never return to using. Formally this means that for any non-trivial subset of states $S' \subset S$ there is a non-zero probability path to a state in $S \setminus S'$.

The entropy rate of a stochastic process χ is

$$H_{rate}(\chi) = \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2, \ldots X_n)$$

when the limit exists. Cover and Thomas prove in [CT06] chapter 4 that for a stationary stochastic process the limit exists.

For stationary and ergodic processes we have the following result, which is proved for a finite-valued alphabet in chapter 16.8, theorem 16.179 (Shannon-McMillan-Breiman theorem) of [CT06]

$$-\frac{1}{n} \log p(X_1^n) \xrightarrow{n\to\infty} H_{rate}(\chi), \text{ with probability 1.}$$

We can now approximate the entropy of the process with a sufficiently large sequence $D = (X_i)_{i=1}^n, X_i \sim p(x)$ sampled from the process.

$$H_{rate}(D) \approx -\frac{1}{n} \log p(x_1^n)$$

And similarly when we use the artificial distribution $p^M(x_1^n)$ of the language model M we can approximate the entropy rate as

$$H(D, p^M) \approx -\frac{1}{n} \log p^M(x_1^n)$$

This is the equation we use when defining the entropy of a language model on a test corpus D.

Perplexity (ppl) is entropy exponentiated. In the previous calculations we used the base 2 logarithm, but when using perplexity it does not matter which base of the logarithm we use. The exponentiation of positive bases preserves the relations between the exponents. Therefore comparing perplexities is the same as comparing entropies. In the NLP community perplexity is usually used instead of entropy. The entropy rate and perplexity of an observation is

$$\begin{aligned} H(D, p^M) &\approx -\frac{1}{|D|} \log_b(p^M(D))) \\ ppl(D, p^M) &= b^{H(D,p^M)} \\ &= p^M(D)^{-1/|D|}. \end{aligned}$$

The SRILM toolkit, which is used in the experiment section of this thesis, uses base $b = 10$. The SRILM toolkit also ignores zero-probabilities and out-of-vocabulary words when calculating perplexity. In Appendix E there is an example of how SRILM is used to calculate the perplexity of a document.

3.3 Other evaluation measures to consider

In machine learning we can divide the techniques into supervised and unsupervised. Supervised tasks require a sufficient amount of correctly processed training data that the algorithm uses to learn a model. To validate the algorithms results we need also to set aside part of the training data as a validation set. This unseen validation set will then be used to test the results of the algorithm. In addition, it depends on the supervised algorithm how much training data it needs. In this thesis we only consider supervised methods.

One thing to consider is the amount and quality of training data needed. In NLP and machine learning in general it is difficult to come by well annotated and clean data and manually annotating data is expensive and slow. The corpora used in this thesis are all real world data that are somewhat lacking. Unfortunately, often the problem of unclean data can't be completely avoided.

In supervised methods, the quality of the training data is an important factor. The algorithm can't learn from improper training data a model that would perform better than the quality of the training data. In this thesis we will use AlphaSense documents that were segmented into sentences automatically using typesetting information from the original document in addition to the textual information. We trust the segmentation to be correct. The goal is to use this model to segment

another type of documents, that can't be segmented by the current method. Additionally we use a freely available Reuters corpus that is separated into segments. Most of the segments in the Reuters corpus have only one sentence, but some have more. The documents also have some ambiguous structures which makes it difficult to compare the performance between the text types. The datasets are presented in chapter 5.

One practical thing we need to consider is the efficiency of the model. Usually more complex models perform better, however training and using a complicated model can be time and space consuming. The model needs to perform the segmentation efficiently, since there is a continuous stream of documents to the AlphaSense system and each document needs to be processed fast.

Complex models are also prone to overfitting to the training data. The experiments in this thesis were done on very large training sets, which makes the models less vulnerable to overfitting. Another way to avoid overfitting is to use smoothing methods.

3.4 Human performance

We assume that currently humans perform better on segmenting text to sentences, than any language model. Humans are for now better at handling the context and meaning of a text document.

Stevenson and Gaizauskas [SG00] performed a test on human ability to decide sentence boundaries in transcription of BBC's Nine O'Clock News -programme. The dataset consisted of 534 sentences and presented about 50 minutes worth of broadcast news. They presented three subjects the transcripts without any punctuation and converted to upper case; and to three other subjects the text without punctuation, but retaining the upper and lower case.

They note that humans disagree sometimes on the punctuation of sentences, which makes it difficult to evaluate the results. They use the original transcription that was done by a trained transcriber listening to the original broadcast as the expert to compare the other results to. The first group achieved **precision of 0.84-0.93, recall of 0.68-0.78 and F-measure of 0.76-0.82 on all-uppercase text**. The second group performs better on the **mixed-case text with precision of 0.96-0.97, recall of 0.67-0.90 and F-measure of 0.79-0.94**. On fully punctuated text, humans would likely achieve a near perfect score.

They also note that upper- and lower-case information improves the performance of both human and a computational system. Their experiment was inspired by an experiment comparing human and machine performance on sentence internal punctuation by Beeferman et al. [BBL98].

It is, in fact, quite hard to read unpunctuated text. However, there are also cases where for a human it might be hard to disambiguate the boundary but a language model might be able to decipher the boundary. The following example has most of the punctuation in place, but we have a title and a paragraph that are not separated by a period. It is difficult to see where the title ends and paragraph begins.

> ...Strike Looming - Union Members Reject Stillwater's Offer In what will likely pave the way for a strike, this morning union employees at the company's Stillwater Mine and Columbus processing ...

What the language model sees is that we have a capitalized sequence of words, after which uncapitalized words. It is not far-fetched to expect a language model to pick up on this kind of pattern if there are enough similar instances in the training data. The tokenization of the text is addressed in Chapter 5.2.

> ... <c> strike <c> looming - <c> union <c> members <c> reject <c> stillwaters <c> offer <c> in what will likely pave the way for a strike , this morning union employees at the companys <c> stillwater <c> mine and <c> columbus processing ...

The true title separated from the start of the paragraph

> ...
> Strike Looming - Union Members Reject Stillwater's Offer
> In what will likely pave the way for a strike, this morning union employees at the company's Stillwater Mine and Columbus processing ...

4 The n-gram model

The Markov Model was first introduced by Markov in 1913 to model word sequences. In this model we make an assumption (also know as the Markov assumption) that

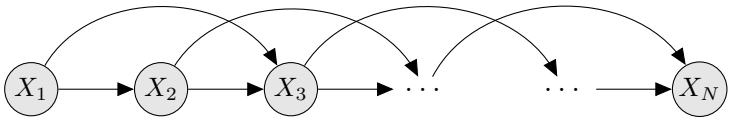

Figure 2: 2nd order Markov model, trigram model. An arrow $X_i \rightarrow X_j$ denotes that the variable X_j is dependent on X_i

the probability of an event, in this case a word, is only dependent on the previous n-1 words, instead of all the words in the document. This frees us to consider an event only in the local context of the word instead of considering the full text sequence. The n-gram model is a time-invariant Markov chain of order $n-1$. The Markov model and maximum likelihood parameter estimation for the n-gram model are presented in Section 4.1. Smoothing methods for the model parameters are discussed in Section 4.2.

Stolcke and Shriberg [SS96] present the hidden segment n-gram model for segmenting conversational speech transcripts. Their approach is based on the n-gram language model. In Section 4.3 we will explain how the n-gram model can be used to segment text using a Viterbi decoding algorithm.

4.1 The Markov Model

Consider the training data as a sequence of states w_i, where each state i corresponds to the $i'th$ word in the sequence. We mark the number of words in the training sequence as N. In practice, we consider also punctuation and different types of tags as words. The probability of a word is only dependent on the n previous words. We also assume that the model is a time invariant. Formally this is expressed as

$$p(w_i|w_1, w_2, \ldots, w_{i-1}) = p(w_i|w_{i-n}, w_{i-n+1}, ..., w_{i-1}).$$

Probability of a sequence of words $D = w_1...w_N$, under the Markov assumption, is

$$\begin{aligned}
p(D) =& p(w_1...w_N) \\
& \text{Chain rule of probability} \Rightarrow \\
=& p(w_1)p(w_2|w_1)p(w_3|w_1w_2)\cdots p(w_N|w_1\dots w_{N-1}) \\
=& \prod_{i=1}^{N} p(w_i|w_1\dots w_{i-1}) \\
& \text{Markov assumption} \Rightarrow \\
=& p(w_1)p(w_2|w_1)p(w_3|w_1w_2)\cdots p(w_k|w_{k-n}\dots w_{k-1})\cdots p(w_N|w_{N-n}\dots w_{N-1}) \\
=& \prod_{i=1}^{n} p(w_i|w_{i-n}\dots w_{i-1})
\end{aligned}$$

The maximum likelihood estimation of the transition probability from word sequence w_i^{i+n-1} to word w_n is based on the counts of matching sequences found in the training corpus

$$p_{ML}(w_n|w_1^{n-1}) = \frac{|w_1^n|}{|w_1^{n-1}|}.$$

We will prove the maximum likelihood estimate for the case of order n=2. The proof is similar for cases $n > 2$.

Proof. Assume we have a vocabulary V, of size $|V|$. The vocabulary is a list of all unique words in the training corpus. Denote the j'th word in the vocabulary as v_j where $j = 1, \dots, |V|$. We call the probability of the word v_k to be followed by the word v_l the transition probability and denote it as

$$\theta_{kl} := p(v_l|v_k).$$

The observation sequence (the training corpus) is a sequence of random variables $D = (W_1 \dots W_N)$ where each $W_i : V \to \mathbb{R}$ is a discrete random variable. Note that we assume the Markov chain to be time invariant, which means that the transition probability from word w_k to w_l does not depend on its place in the sequence. That is, for all $t \in \mathbb{N}$

$$P(W_i = v_l|W_{i-1} = v_k) = P(W_{i+t} = v_l|W_{i+t-1} = v_k) = p(v_l|v_k) = \theta_{kl}.$$

Note that here we use three different notations that refer to a word. We model an observation vector D as a sequence of random variables where W_i is the i'th random variable in the sequence. The concrete observed i'th value of a observation

sequence is w_i. The concrete observation w_i takes a value from the vocabulary V. The vocabulary is the set of words that appear in the corpora and v_j is the j'th word in the vocabulary.

Denote the count of times the sequence $v_k v_l$ occurs in the observations as

$$N_{kl} := |v_k v_l|$$

The likelihood function of the model parameters is

$$\begin{aligned} L(\theta; D) &= p(D|\theta) \\ &= \prod_{i=1}^{N} p(W_i = v_l | W_{i-1} = v_k) \\ &\text{(time invariance)} \\ &= \prod_{l=1}^{|V|} \prod_{k=1}^{|V|} \theta_{kl}^{N_{kl}}, \end{aligned}$$

and the log likelihood function is

$$l(\theta; D) = \sum_{l=1}^{|V|} \sum_{k=1}^{|V|} N_{kl} \log \theta_{kl}.$$

The transition probabilities θ_{kl} have to satisfy the condition

$$\sum_{l=1}^{|V|} \theta_{kl} = 1,$$

since the sum of the transition probabilities from state v_k to any state v_l where $v_l \in V$ must sum up to one to define a proper probability distribution. This must be satisfied for all v_k where $v_k \in V$. This yields $|V|$ conditions. We use corresponding Lagrange multipliers λ_k where $k = 1, 2, ..., |V|$ to take the conditions into account. The target function is now

$$l^*(\theta; D) = \sum_{l=1}^{|V|} \sum_{k=1}^{|V|} N_{kl} \log \theta_{kl} - \sum_{k=1}^{|V|} \lambda_k (\sum_{l=1}^{|V|} \theta_{kl} - 1)$$

The maximum likelihood parameters θ are found by solving the root of the derivate

of the target function:

$$\begin{cases} \frac{\partial l^*}{\partial \theta_{kl}} = N_{kl}\frac{1}{\theta_{kl}} - \lambda_k = 0 \\ \frac{\partial l^*}{\partial \lambda_k} = \sum_{l'=1}^{|V|} \theta_{kl'} - 1 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} \theta_{kl} & = \frac{N_{kl}}{\lambda_k} \\ \sum_{l'=1}^{|V|} \theta_{kl'} & = 1 \end{cases}$$

$$\Rightarrow \sum_{l'=1}^{|V|} \frac{N_{kl'}}{\lambda_k} = 1 \Leftrightarrow \sum_{l=1'}^{|V|} N_{kl'} = \lambda_k$$

$$\Rightarrow \theta_{kl} = \frac{N_{kl}}{\sum_{l'=1}^{|V|} N_{kl'}} = \frac{|v_k v_l|}{|v_k|}$$

□

The reasoning in case $n > 2$ is exactly the same, but instead of preceding word $v_k \in V$ we consider the sequence of words $v_{l-n} \dots v_{l-1}$ preceding the word v_l.

4.2 The back-off model and smoothing techniques

When using maximum likelihood estimates we quickly notice a problem: if we have not seen an (n-gram) event in the training set, we consider its probability to be zero. Since probabilities are calculated by multiplicating independent probabilities, even one zero probability causes the total probability of a sequence to be zero, no matter how probable we consider the rest of the events. In a smoothing technique we reallocate some of the probability mass of the seen events (n-grams in training) into the unseen events, so that no event has a zero probability.

Another problem we note is that the event space in a natural language is huge: an n-gram model has

$$\sum_{i=1}^{n} |\text{words in vocabulary}|^i$$

events. Not all of these events are present in training. However, the model needs to be able to assign a larger than zero probability for each possible event. In this chapter we will consider smoothing using additive smoothing and backing off and interpolation.

Additive smoothing, also known as Laplace smoothing (1812) is possibly the simplest smoothing method available. All probabilities are smoothed by

$$p^*(w_n|w_1...w_{n-1}) = \frac{|w_1..w_n| + \alpha}{|w_1..w_{n-1}| + \alpha|V|.}$$

This gives all events a probability that is larger than zero and the conditional probabilities still sum up to one. If $\alpha = 1$ this corresponds to considering that each n-gram appeared one more time than it did.

As mentioned, the event space in a natural language is huge, and there are significantly more n-grams that have not been observed, compared to n-grams that were observed. This smoothing method assigns large amount of the probability mass to unseen events, making the probabilities of the observed events significantly smaller. In light of this, additive smoothing is considered to be an inadequate smoothing method for n-gram models. However, due to its simplicity it is usually presented to explain the idea of smoothing.

In this chapter we will define the back-off and interpolated discounting methods. The discounting method used in most of the experiments in this thesis, is the Witten-Bell discounting method. We will present both the back-off version and the original interpolated version of Witten-Bell discounting. There are also a large array of other discounting methods that follow the same idea of back-off or interpolation.

The back-off model was first introduced by Katz [Kat87] for Good-Turing smoothing. The idea of backing-off is that when encountering an unknown n-gram the model backs off to approximating the n-gram probability with a lower order gram. To do this, we take (discount) some probability mass from the n-gram and reserve it for the backing-off. We denote the back-off weight of $(n-1)$-gram w_1^{n-1} as $bow(w_1^{n-1})$. The back-off n-gram model's probabilities p_{BO} are expressed as

$$p_{BO}(w_n|w_1^{n-1}) = \begin{cases} p^*(w_n|w_1^{n-1}) & \text{, if n-gram was seen during training} \\ p_{BO}(\underbrace{w_n|w_2^{n-1}}_{(n-1)-gram}) * bow(\underbrace{w_1^{n-1}}_{(n-1)-gram}) & \text{, for unseen n-gram.} \end{cases}$$

Here the p^* is the discounted probability assigned to a n-gram that was seen during training.

The back-off version of Witten-Bell Next we will present the back-off version of Witten-Bell discounting. The closer motivation for defining the equations is

discussed when presenting the interpolated version. Let $N(w_1^{n-1}*)$ be the count of unique words that follow the $(n-1)$-gram w_1^{n-1} in the training sequence.The discounted probabilities and back-off weights are defined as

$$N(w_1^{n-1}*) = |\{w_n : |w_1 \dots w_{n-1} w_n| > 0\}|$$

$$p^*_{WB}(w_n|w_1^{n-1}) = \frac{|w_1^n|}{N(w_1^{n-1}*) + |w_1^{n-1}|}$$

$$bow_{WB}(w_1^{n-1}) = \frac{1 - \sum\limits_{w_n : |w_1^{n-1} w_n| > 0} p^*_{WB}(w_n|w_1^{n-1})}{1 - \sum\limits_{w_n : |w_1^{n-1} w_n| > 0} p^*_{WB}(w_n|w_2^{n-1})}$$

The back-off Witten-Bell probabilities are expressed recursively as

$$p_{WB}(w_n|w_1^{n-1}) = \begin{cases} p^*_{WB}(w_n|w_1^{n-1}) & \text{, if n-gram was seen during training} \\ p_{WB}(w_n|w_2^{n-1}) * bow_{WB}(w_1^{n-1}) & \text{, for unseen n-gram.} \end{cases}$$

The first step in understanding the back-off model is to notice that the discounted probability p^*_{WB} of an n-gram is smaller than the maximum likelihood estimation p_{ML}. When $N(w_1^{n-1}*) > 0$, the following equation holds

$$\frac{|w_1^n|}{N(w_1^{n-1}*) + |w_1^{n-1}|} < \frac{|w_1^n|}{|w_1^{n-1}|}$$

$$p^*_{WB}(w_n|w_1^{n-1}) < p_{ML}(w_n|w_1^{n-1})$$

We note that the probability mass reserved for cases when an unseen word w_n appears after the w_1^{n-1} $(n-1)$-gram, is one minus the sum of probabilities of all observed n-grams that begin with the prefix w_1^{n-1}

$$1 - \sum_{w_n \in V : |w_1^{n-1} w_n| > 0} p^*_{WB}(w_n|w_1^{n-1}).$$

This leftover probability mass is distributed to the zero-count cases, using normalizing back-off weights (bow) and lower order n-grams w_2^n, so that the probability of an unseen n-gram is calculated as

$$p_{WB}(w_n|w_1^{n-1}) = bow(w_1^{n-1}) * p_{WB}(w_n|w_2^{n-1})$$

We need to make sure the conditional probabilities of $w_n \in V$, given w_1^{n-1} for both unseen and seen events, sum up to 1

$$\sum_{w_n \in V} p_{WB}(w_n|w_1^{n-1}) = \sum_{w_n : |w_1^n| > 0} p^*_{WB}(w_n|w_1^{n-1}) + bow(w_1^{n-1}) * \sum_{w_n : |w_1^n| = 0} p_{WB}(w_n|w_2^{n-1})$$

$$= 1$$

This is achieved when the back-off weight of each w_1^{n-1} is normalized in the following way

$$bow(w_1^{n-1}) = \frac{1 - \sum_{w_n:|w_1^n|>0} p^*_{WB}(w_n|w_1^{n-1})}{\sum_{w_n:|w_1^n|=0} p_{WB}(w_n|w_2^{n-1})} = \frac{1 - \sum_{w_n:|w_1^n|>0} p^*_{WB}(w_n|w_1^{n-1})}{1 - \sum_{w_n:|w_1^n|>0} p^*_{WB}(w_n|w_2^{n-1}).}$$

Interpolated discounting methods always consult the lower order n-grams, not just in cases when the n-gram's count is zero. The Witten-Bell discounting method is originally an interpolated discounting method. The interpolated probabilities are expressed recursively as

$$p_{IP}(w_n|w_1^{n-1}) = \lambda_{w_1^{n-1}} p_{ML}(w_n|w_1^{n-1}) + (1 - \lambda_{w_1^{n-1}}) p_{IP}(w_n|w_2^{n-1}),$$

where $0 \leq \lambda \leq 1$ is a variable that controls the amount of interpolation to the lower order model. The factor $(1 - \lambda_{w_1^{n-1}})$ can be thought of as a weight on how much we want to consult the lower order n-gram, instead of the maximum likelihood estimation. This is similar to the back-off weight in the back-off model.

Witten-Bell discounting is an interpolated discounting method presented by Bell et al. in [WB91] as method C. The idea of Witten-Bell discounting is that, if we observe a word is followed by a large number of unique words in training, it is likely to be followed by even more unique words in testing. Conversely, if a word is followed only by a handful of unique words, it is less likely to be followed by an unseen word in training. The conclusion is that we should discount more probability mass to the unseen events in the first case than in the second. We also take into account the number of times the first word appears in training. Words that appear often are more likely to be followed by several unique words, than words that already only appear a couple of times. We define the lambdas of Witten-Bell discounting as

$$1 - \lambda_{w_1^{n-1}} = \frac{N(w_1^{n-1}*)}{N(w_1^{n-1}*) + |w_1^{n-1}|}.$$

The Witten-Bell interpolated probabilities are expressed as

$$p^*_{WB}(w_n|w_1^{n-1}) = \lambda_{w_1^{n-1}} p_{ML}(w_n|w_1^{n-1}) + (1 - \lambda_{w_1^{n-1}}) p^*_{WB}(w_n|w_2^{n-1}).$$

We present an example on 2-grams using the Reuters corpus that will be introduced in Chapter 5. The higher order models work the same way, but instead of a word we consider a $(n-1)$ sequence of words, and how many times that particular sequence was followed by a new word. The word "any" and "investment" both appear exactly 1406 times in the training data. However, the word "any" is followed by 1200 unique words; and "investment" by only 236. The corresponding lambdas are

$$1 - \lambda_{any} = \frac{1200}{1200 + 1406} \approx 0.460$$
$$1 - \lambda_{investment} = \frac{236}{236 + 1406} \approx 0.144.$$

When there are relatively few unique words followed by the word, we trust the maximum likelihood model more, and the $(1-\lambda)$ is smaller; and in the case when there are relatively many unique following the word, we take the lower order into account more and the $(1-\lambda)$ is larger. In the bigram case, the smaller the lambda is, the more we consult the 1-gram distribution of the following word.

In the case of Witten-Bell discounting, $0 < (1-\lambda) \leq 0.5$. The case $(1-\lambda) = 0.5$ occurs when the word is always followed by a unique word. If the word is followed only by one unique word, the $(1-\lambda)$ is small, but gets larger if the occurrences of the words are few. For example, for a word that only appears once both $N(w*)$ and $|w|$ equal to 1 and λ is therefore equal to 0.5.

Other discounting methods include Good-Turing Estimate (1953), Ristad's natural discounting (1995), Kneser-Ney and Modified Kneser Ney. Chen and Goodman [CG96] give a good overview of these discounting methods. The Kneser-Ney discounting methods were considered to be the best discounting methods by Chen and Goodman. They were, however, excluded from the experiments in this thesis because of unexpectedly bad results using the SRILM toolkit. Different discounting methods are compared briefly in Chapter 6.2. In the experiments by Chen and Goodman the differences in perplexity of the different discounting methods became smaller with larger training sets.

Stolcke noted in [S+02] that the interpolated versions of discounting methods gave slightly better results. In Chapter 6.4 we will compare the back-off and the interpolated models in regards of perplexity and F-score. In our experiments we note that the interpolated model achieves slightly better results.

The SRILM toolkit that is used in the experiment section of this thesis uses an format called ARPA to store the back-off and interpolated models. The ARPA

format is explained in Appendix B, and an example of creating a language model using SRILM is presented in Appendix C. An example of using the ARPA format back-off model to calculate the probability of a sentence is presented in Appendix D.

4.3 Viterbi algorithm for hidden n-gram

Suppose we have trained an n-gram language model using data that has sentence boundary-tokens <B> in place. This means we have defined the probability for all n-grams in the vocabulary. This includes all possible n-grams that contain the boundary token. We are now presented with a new document that does not have any boundary tokens in place and we need to decide where boundary tokens should be placed.

The Viterbi algorithm is best known as an algorithm used for finding most probable hidden events in output of a Hidden Markov Model. The Viterbi decoder is also a more general term used to describe a dynamic programming algorithm that finds an event-path that maximizes the probability of the events of a chain structured model. We now present a Viterbi decoding algorithm that decides where to place boundary tokens in a word sequence.

Dynamic programming algorithms are also referred to as trellis algorithms in the field of language modeling. The trellis refers to the way that a dynamic algorithm builds on the results of the previous step.

We take the observation sequence, and add a possible unseen event e_t between each word. The e_t is either <B>, or a no-event. The no-event corresponds to not having any event at that index. The same algorithm can be applied to a different type of problem by defining a different set of hidden events. The different hidden classes were discussed in Chapter 2.2. The algorithm maximizes the joint probability

$$p(w_1^T, e_1^T) = p(w_1, e_1, w_2, e_2, \ldots w_T, e_T)$$

of the sequence.

The Viterbi algorithm is based on the observation that the most probable path ending in hidden event s_j at time index t must follow from a most probable path to some state at time index $t-1$. Denote the probability of event e_t being $s_j \in \{< B >, no-event\}$ as $\delta_j(t)$, when being preceded by the most probable path. Let K be the number of hidden states, and T the number of elements in the sequence.

Denote the transition probability from the n-gram $x = w_i \dots w_{i+n}$ to the word w_y, resulting in state $w_{i+1} \dots w_{i+n} w_y$, as θ_{xw_y}.

The algorithm proceeds as follows:

- For the initialization step
  - Initialize the probability for each hidden state s_j at event e_1
    $\delta_j(1) = p(s_j|w_1)$
  - Initialize the back-trace as
    $\psi_j(1) = w_1 s_j$
- For each step $t = 2, \dots, T$:
  - For each backtrace at $t-1$, calculate the transition probability to the word w_t and the resulting n-gram

$$\delta'_j = \delta_j(t-1)\theta_{\psi_j(t-1)w_t}$$
$$\psi'_j = \psi_j(t-1)w_t$$

  - For each hidden state s_j, assign $\delta_j(t)$ to be the probability of most probable path from an n-gram in the backtrace to the hidden state s_j

$$\delta_j(t) = \max_{i=1\dots K} \delta'_i \theta_{\psi'_i s_j}$$

  - For each hidden state s_j, store the backtrace i.e. the transition that maximized the probability

$$\psi_j(t) = \arg \max_{\substack{\psi'_i s_j: \\ i=1\dots K}} \delta'_i \theta_{\psi'_i s_j}$$

- Finally, take the probability at index T that maximizes the probability of the full sequence, and trace back the path that leads to it.

The time complexity of the algorithm is $O(K^2T)$ and space complexity $O(KT)$. The SRILM toolkit has an implementation of the Viterbi algorithm under the function hidden-ngram. An example of segmenting using the Viterbi algorithm is presented in Appendix F.

5 The datasets

In this section we will present the datasets that will be used in experiments with the n-gram models in Chapter 6. The documents are fully punctuated and separated into sentences. The AlphaSense documents that we use were segmented automatically by the current system that uses additional typesetting information from the original document in addition to the plain text. In some cases the punctuation is missing from titles and there are other ambiguous structures such as address lines, bullet points, data tables and so on.

In Subsection 5.1 we will present the different types of text types that the datasets consist of. In Subsection 5.2 we will address how the segmented data was separated and tagged into tokens before learning a language model. In Subsection 5.3 we compare the vocabularies and sizes of the datasets to give an overall picture of the similarities and differences of the sets.

5.1 The datasets

In this section we will present the datasets that are used in the experiments in Chapter 6. We will further compare the sizes and vocabularies of the datasets in Subsection 5.3. Most of the text documents from the AlphaSense system have been extracted from an original document that contained the text.

The **A**, **B** and **C** datasets are document types from the AlphaSense system. The documents are all different types of news stories that cover financial and economic subjects. The documents were segmented automatically by the AlphaSense pipeline with additional information from the original document.

Reuters-21578 Distribution 1.0 (R)[2] is a dataset that was collected and labeled by Carnegie Group, Inc. and Reuters, Ltd. The dataset is freely available for research purposes. The dataset consists of news stories that were published in the Thompson Reuters newswire in 1987. This dataset is used as reference to compare the segmentation performed by other models, to provide context for the reader. The dataset is significantly smaller than the AlphaSense document-sets.

The Reuters corpus is segmented into paragraphs. Almost all of the paragraphs contain only one sentence and are therefore the corpus was deemed good enough as

[2]http://www.daviddlewis.com/resources/testcollections/reuters21578

a reference corpus. Unlike the AlphaSense documents, the Reuters corpus' segments contain complete sentences, whereas the AlphaSense documents might contain false sentence boundaries. Despite the failings of the Reuters corpus it was the best freely available corpus found.

The **D** dataset is the dataset we would like to learn to segment. Unlike the A, B and C -document types, we don't have an effective way to segment the D documents. For the experiments in Chapter 6, some documents were run trough a slow conversion process so that the documents could be segmented using the same methods as the A, B and C documents. The conversion process is too slow to use at runtime, hence the documents can't be segmented using this method in the AlphaSense product. However, this way we gained some automatically segmented documents for testing the language models. The 3000-document sample dataset is not used in training, only as a test set.

The MIX dataset is the joint set of the A, B and C -sets. The MIX set is used to learn a language model in Chapter 6 and compared to the performance of the language models learned only on one type of data. The joint vocabulary is compared to the other vocabularies in Subsection 5.3.

5.2 Preprocessing the data

Most real world data has a lot of "garbage" that is unnecessary when considering only the sentence segmentation task. These include meta-data such as headers, or other lexically uninportant parts such as data tables. Many of the data also include an identical or near-identical copyright statement which might trick the algorithm into giving too much weight to certain word-sequences that might not model the rest of the data.

First step in the preparing the data is to remove metadata, tables and other parts of the document that are unimportant for the segmentation task. This is an important step, as the clutter of unnecessary data can disturb training and applying a language model. In practice it is not possible to remove all clutter automatically.

As mentioned before, the datasets have some ambiguous structures that can't all be cleaned out automatically. This causes some ambiguities in the text types: it might be that in one type an email and postal address are always divided into two or three separate sentences; and in another type they might be as one sentence. This makes comparing the segmentation hard, as there might be several correct ways to

separate them into sentences. This is a problem that can't be fully avoided when working with real world data.

The capitalization information of a word is especially important for the segmentation task: sentences usually start with a capital letter, so do names and the pronoun I. Many company names and abbreviations are in full capital. To include the case information in the language model, we add a tag-token in front of all words that contain capital letters. After this we convert everything to lower-case. The tokens we use are:

- <fc> for word that is full in uppercase
  LA, NBC, and sometimes a title is in full capital
- <c> for word that begins with capital letter
  October, French
- <sc> for word that is a single-letter capital letter
  I, and abbreviations separated into tokens: U . S .
- <cc> for word that contains a capital letter that is not the first letter
  iPhone, d'Automatismes, iHeartCommunications
- <c><cc> for word that begins with a capital letter and is in CamelCase
  AlphaSense, TeliaSonera, WhatsApp

The financial documents contain a lot of unique numbers: telephone numbers, percents, monetary values and so on. The exact number rarely matters for the language model, and the same number rarely appears across the training set. To take this into account we replace all digits with the tag <d>.

All foreign symbols are removed from the text. Only tags, letters a-z and symbols , " ( )| . / # - : ; % * $? ! & + = are retained. In Section 6.5 we perform a small test on retaining different sets of symbols. A space is added between all symbols to separate them into tokens. This causes also some abbreviations to be separated into different tokens if they contain a period.

The original documents are formatted so that each line has one sentence. As a last step we tag each sentence break with a <B> and remove the line breaks. This is done because of the SRILM toolkit. The SRILM toolkit considers each line as a separate segment. If we keep each sentence as their own segment, we can't model the n-grams that have the sentence break in the middle of the n-gram, since the tag only appears at the end of the sentence.

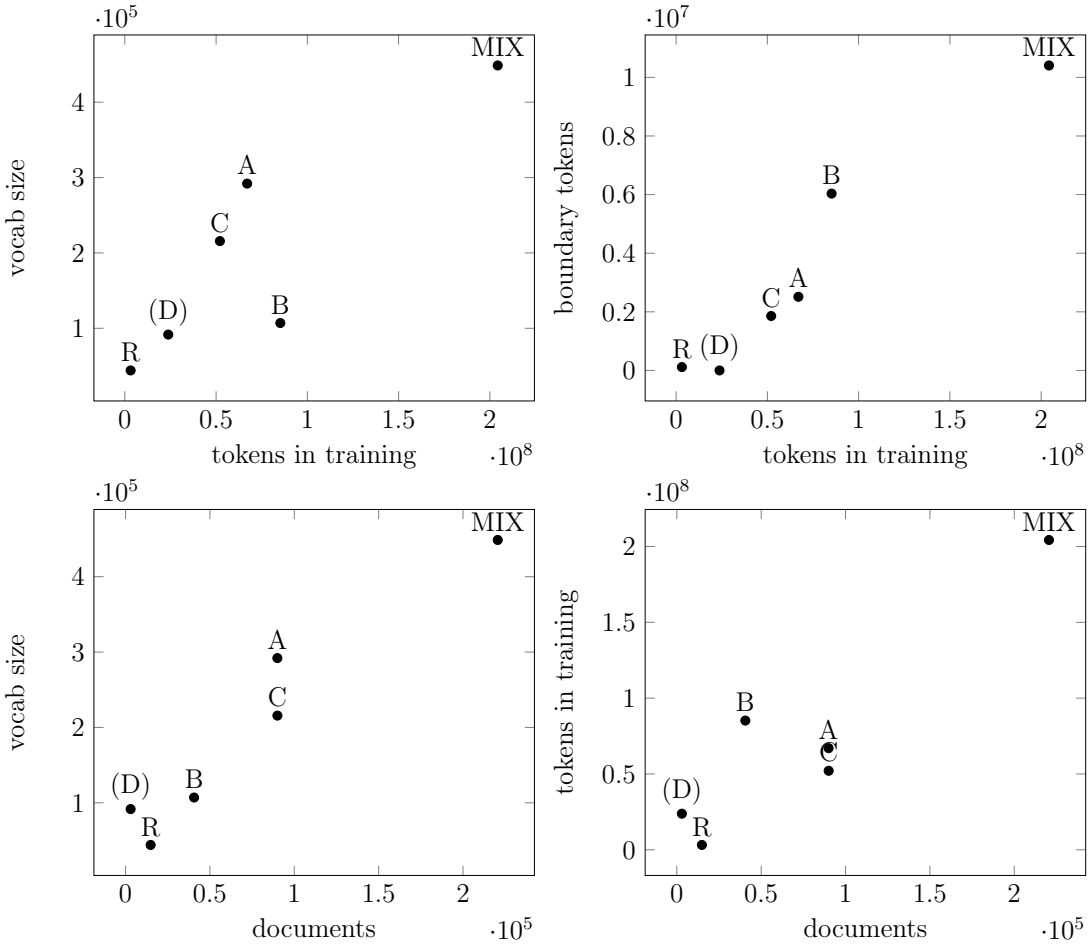

Figure 3: Comparison of the training datasets A, B, C and R, and the set D, in regards of vocabulary size, tokens, boundary tokens and documents

5.3 Comparison of vocabularies

In this section we will take an look on the differences on the size and content of vocabularies of the datasets. We are especially interested in comparing the other datasets to the dataset D. The datasets were introduced in Section 5.1.

The plots in Figure 3 compare the sizes and tokens of the datasets. Token refers to either a word, or another token such as <B> or <c>. Boundary token is the <B> token that separates sentences.

The D and R datasets have significantly fewer documents and tokens in the dataset. As mentioned, this is because the R is a smaller dataset and D is a smaller testing set. The MIX dataset is the joint set of A, C and B, and has therefore the sum of their documents and tokens in training.

comm(a,b)/a

a\b	A	B	C	R	D	MIX
A	1.0	0.17912	0.35708	0.10495	0.14639	1.0
B	0.48881	1.0	0.50615	0.22618	0.33174	1.0
C	0.48335	0.25107	1.0	0.13990	0.21036	1.0
R	0.69539	0.54919	0.68481	1.0	0.52146	0.76516
D	0.46637	0.38727	0.49507	0.25071	1.0	0.56893
MIX	0.65065	0.23843	0.48067	0.07513	0.11620	1.0

Figure 4: Proportion of the vocabulary of the type on the row that is also in the vocabulary of the type on the column

The vocabulary size grows as the number of tokens grows. The only anomaly is the B set that has a significantly smaller vocabulary, even though it has more tokens than the C and A sets. This might be a result of near identical documents, repetition or some other problem in the dataset. However, the B vocabulary is more in line with the others when comparing the vocabulary size to number of documents. This suggests there is either repetition in the content of documents or in some part in all the documents that was not caught in the preprocessing steps.

For the datasets R, C and A the relation of boundary tokens to all tokens is about 0.036 ± 0.001, and for B 0.0708. This means there are significantly more boundary tokens in B relatively than in the other types. This again speaks for the abnormality of the B set. The D set does not have any boundary tokens.

From the documents and tokens comparison we can note that the datasets R, A, C have relatively shorter documents than the B and D sets. Especially the B set has more tokens, even though it has fewer files.

To get a better picture of the vocabularies we compare them to each other. Figure 4 has a comparison of the proportions of words in the vocabulary of one type contained in the vocabulary of another type. Read the table as: size of the intersection of the two vocabularies, divided by the size of the vocabulary on the row. This is the proportion of the vocabulary of the type on the row that is also in the vocabulary of the type on the column.

The A type has the largest vocabulary so its vocabulary contains about half of all of the other vocabularies. Interestingly this is about as good as the intersection gets. This suggests that about half of the words in the vocabularies are words that occur

1000 most common words comparison

	A	B	C	R	D
A	1000	442	689	639	522
B	442	1000	532	506	564
C	689	532	1000	685	635
R	639	506	685	1000	566
D	522	564	635	566	1000

1000 most common 2-grams

	A	B	C	R	D
A	1000	264	446	448	290
B	264	1000	349	287	374
C	446	349	1000	448	455
R	448	287	448	1000	334
D	290	374	455	334	1000

1000 most common 3-grams

	A	B	C	R	D
A	1000	119	291	268	127
B	119	1000	194	153	167
C	291	194	1000	248	254
R	268	153	248	1000	150
D	127	167	254	150	1000

1000 most common 4-grams

	A	B	C	R	D
A	1000	50	170	170	54
B	50	1000	105	82	82
C	170	105	1000	154	162
R	170	82	154	1000	81
D	54	82	162	81	1000

Figure 5: Comparison of the size of the intersection of the 1000 most common words of each vocabulary

rarely, or are specific to the text type. The R set has a very small vocabulary, and is therefore less likely to have a significant amount of rare words, but still at least 23% of its words aren't contained in the other vocabularies. This is an important notion, as the n-gram model used in this thesis is highly dependent on the vocabulary. When the model is presented to a text type it has few words in common with, it can't be expected to work well. The A and C types seem the most similar to the D and R types.

The previous comparison was biased because of the different sizes of the vocabularies. Another factor is the times the words appear: it might be almost 50% of the words that were unique to the text type only appeared a couple of times in the whole set and it is only by chance they didn't appear in the other sets. The previous comparison also ignored the sequential features of the words. The tables in Figure 5 compare the intersections of the 1000 most common words and n-grams that appeared in the vocabularies.

The intersection of 1000 most common words in vocabularies follows almost the same pattern: 30%-50% of the words are not included in the other vocabularies' 1000 most common words. There is also more diversity in the vocabularies the

a\b	A	B	C	R	D
A	1000	865	956	760	404
B	465	1000	559	308	328
C	626	736	1000	374	473
R	575	585	652	1000	381
D	607	863	813	347	1000

Figure 6: Intersection size of 1000 most common 4-grams of type a against full vocabulary of type b

higher order n-grams we compare. The R set is similar to A and C, and the D set is similar to the B and C set.

We note that this comparison is biased on the strict decision to use only the 1000 most common words. It might be the 1000 most common words are contained in the other vocabulary, but didn't make it to its top 1000 words. Finally in Figure 6 we compare the 1000 most common words of one vocabulary to the full vocabulary of the other type. We only compare the 4-grams, as the most common lower order n-grams were almost always all included in the other text type's full vocabulary.

This test is again biased with the vocabulary size. In the A set, whose vocabulary is largest, the most common 4-grams seem to present a good sample of most common 4-grams, as they are well contained in the other type's vocabularies. The R set's common 4-grams are contained relatively well in all A, B and C. The D set stays most similar to B and C.

6 Experiments with n-gram models and sentence segmentation

There are several variables that affect the performance of any language model. We need to decide how the training data will be cleaned for training, and how much training data is enough. In the case of n-gram models we need to decide on the order of the model and the smoothing method used. Since our goal is to learn a language model on one type of data and use it to segment the D documents, we also wonder how well the language models trained on one type of data adapt to segmenting the data from other text types.

In this chapter we will present some empirical results than shed light to these questions. In Subsection 6.2 we will experiment on the R dataset how the order and smoothing method affect the model's perplexity and performance on the segmentation task. After this chapter the tests are done with a 4-gram model that is smoothed using the Witten-Bell smoothing method.

In Subsection 6.3 we will compare how the language models trained on different AlphaSense document types perform on the test sets of other document types. This also ties to Chapter 5.3, where we compared the vocabularies of the different text types. We will also compare how using the interpolated version of Witten-Bell smoothing improves the performance of the model. The observations are presented Subsection 6.4.

Additionally we will test whether retaining different punctuation and symbols on C training data documents can improve the segmentation of R and D documents. The results are presented in Subsection 6.5. We will also test in Subsection 6.6 how the number of C documents used in training of the model affects the performance on segmenting D and R documents.

The experiments are done using the SRI language modeling toolkit (SRILM). SRILM has a wide range of smoothing techniques and other tools. More information about the toolkit can be found in Appendix A. There is also information in the Appendixes on how the toolkit can be used to build an n-gram model and to segment text with the Viterbi algorithm.

All datasets are divided into two separate sets: a training set used to train the models; and an unseen testing set used to evaluate the performance of the model. Both sets are tokenized in the same way so that the language model is compatible to the testing set. The tokenization of text was addressed in chapter 5.2.

Measuring the perplexity of a model was addressed in Chapter 3.2, and the measures F-score, precision and recall in Chapter 3.1. The segmentation was done with the Viterbi algorithm implementation of SRILM. The theory of the algorithm was presented in Chapter 4.3.

6.1 Results from experiments in literature

The n-gram model has been used especially for segmenting speech transcripts. The problem setting is different from the case where we have the capitalization information and punctuation. However, we can use these as reference for the order of the

model and differences between interpolated and back-off models. We can also get clues on how well an n-gram model might perform in cases where the punctuation is missing from a title.

Stolcke and Shriberg [SS96] present the hidden segment model for segmenting conversational speech transcripts. Their 3-gram model achieves recall of 0.702 and precision of 0.607 on plain words with no punctuation or case information. Gotoh and Renals [GoS00] used n-gram models of $n = 2, 3, 4, 5$ in sentence boundary detection for broadcast speech transcripts. They used data that was converted fully to lower case and no punctuation at all. In their experiments the 3-gram model performed significantly better than the 2-gram model. Higher order n-grams did not improve the performance after this.

Stolcke and Shriberg also compared back-off and interpolated models but did not note any significant difference in their performance. They also tested how the training set size affects the performance and noted that the performance improves as the training set grows. Their lexical language model achieved recall of 0.39 and precision of 0.56.

In our experiments we have converted all text to lower case, but have added a case-information token before each word that contains upper case letters. In most cases we have retained the punctuation of the text. The case information tokens are presented in Chapter 5.2.

The most important and extensive test in this thesis is in Subsection 6.3 where we test how the language models trained on one type of data perform on another type of text.

6.2 Smoothing and order of model

In this section we will test how the smoothing method and order of the model affect the performance of the n-gram model. Since the AlphaSense documents used for this thesis are not available freely, we will present some results on also on a freely distributed corpora.

The corpus used in this section is the Reuters corpus [3] that is also referred to as R in this thesis. The dataset was divided by a 80/20 split: We use 18583 documents for training (containing 147729 sentences) and 3717 documents for testing.

[3] Reuters-21578 Distribution 1.0
`http://www.daviddlewis.com/resources/testcollections/reuters21578`

Reuters-21578, Distribution 1.0
14866 training files, 3717 testing files; 80/20-split
testing: 784795 words, 6352 Out-Of-Vocabulary words

perplexity	1-gram	2-gram	3-gram	4-gram	5-gram	6-gram
add 1	376	385	823	1344	1760	2047
add 0.01	375	90	75	89	103	113
Ristad's natural discounting	375	74	42	37	36	35
Witten-Bell	376	73	41	37	36	35
Good-Turing	376	85	49	45	44	43

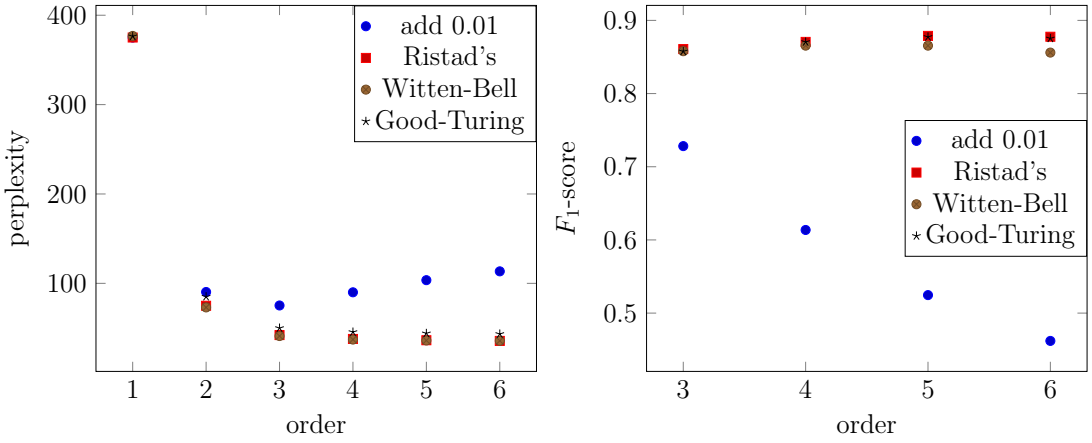

Figure 7: Results of different smoothing techniques on the Reuters (R) corpus

The additive and Witten-Bell smoothing methods were presented in Chapter 4.2. Additionally the Ristrad's natural discounting and Good-Turing implementations of SRILM were included. All the smoothing methods used in this section are the back-off versions. The details of the smoothing methods can be viewed at the manual page ngram-discount of SRILM [4].

In Figure 7 we can see the perplexity on the test set of the different smoothing methods and orders of n-gram models. The same results are shown in the graph on the left; and on the right graph, the corresponding F-measures of the model's performance on the segmentation task. The segmentation results of the models below order 2, and the add-1 smoothing are not included, as these models assigned barely any sentence boundaries.

As expected, the additive smoothing method performs very badly by both the per-

[4] `http://www.speech.sri.com/projects/srilm/manpages/ngram-discount.7.html`

plexity and F-score measures. We can make the conclusion that this simple smoothing method is not of any use in this task. As for the more complicated smoothing methods, they all perform almost equally well. The function ngram-count was used to build the language models in this chapter. For the larger language models in the following chapters we will use the make-big-lm wrapper function that supports only the Witten-Bell, Good-Turing and modified Kneser-Ney smoothing methods. In the experiments in this chapter and other preliminary experiments with the AlphaSense documents, the Witten-Bell proved to be reliable and was therefore chosen for the experiments with the larger datasets.

The perplexity of the model improves as the order of the model grows, except for the additive smoothing method. However, as the order of the model grows, so does the space needed to store the model. This also results in a slower model, as it is slower to fetch the probabilities associated with the n-grams. The improvement in perplexity and F-score are small after the 4-gram models, and the F-score even decreases when comparing the 6-gram model to the 5-gram model. In experiments in literature such as Stolcke and Shriberg [SS96] and Gotoh and Renals [GoS00] the typical order of the model is 3. Since our data has the additional capitalization tokens, a 3-gram model will not always contain as many true words as in their experiments. However, the models that have an order of 5 or more become very large. For these reasons the 4-gram model was chosen for the experiments on the AlphaSense documents.

6.3 Adaptability on other text types

In the previous section we tested how the smoothing method affects the perplexity and performance on the segmentation task of the model. The 4-gram model and Witten-Bell discounting method were chosen for further testing based on the experiments in literature, the experiments in the previous section, and because of practical requirements such as the size of the model and the availability and reliability of the smoothing methods in the make-big-lm wrapper of SRILM. In this chapter we will consider the AlphaSense documents, and how language models trained on one type of documents perform on another type.

In Chapter 5 we presented the different datasets that will be used in the following tests. There are 5 types of language models we train: A, B, C, R, and MIX. The specifics for each training set for these language models are specified in Figure 8. Chapter 5 has further comparison of the training sets and vocabularies of the models and spesifics of the documents.

lm training set	Files in training	sentence boundaries in training	vocabulary size	text types
A	89 965	2 513 198	292 002	A
B	40 611	6 034 764	107 005	B
C	90 000	1 858 279	215 720	C
R	14 866	1 14 970	44 070	R
MIX	220 576	10 406 241	448 736	A, B, C

testing set	number of files	number of sentences
A	3000	85 609
B	3000	262 843
C	3000	61 884
R	3717	28 765
D	3000	834 131

Figure 8: Specifics of the training and testing sets

We are especially interested in how the language models perform on the R and D datasets. The R dataset is a dataset we know is correctly segmented, compared to the other sets that were automatically segmented; and the D documents are the ones we would like to learn to segment.

We will test both a back-off model and an interpolated model, using the Witten-Bell discounting. Both models are 4-gram models. The details of the discounting methods were presented in chaper 4.2. Appendix C has details on how the models are created using the SRILM toolkit. Further discussion on the differences in performance of the back-off and interpolated models is in the next Section 6.4.

The tables in Figure 9 have on the left side results by the models smoothed with back-off, and on the right side the comparison to results given by the model smoothed using interpolation. Each table has on the rows the language models and on the columns the different text types of the testing set. Each cell corresponds to the performance of the language model of the row on the test set of the text type in the column.

The comparison of the vocabularies in Chapter 5.3 indicate some similarities between the text types, but none of the comparisons can be used straightforwardly to predict the results on perplexity or segmentation. This is expected, as the model takes into account the frequency of the words and n-grams, rather than just their existence in

perplexity back-off

lm \text	A	B	C	R	D
A	86	238	160	222	111
B	906	11	332	584	77
C	194	148	27	205	75
R	293	285	247	43	127
MIX	109	13	31	164	48

improvement of interpolated model

lm \text	A	B	C	R	D
A	-9	-15	-13	-24	-2
B	0	-5	+3	+12	-2
C	-4	0	-8	-4	0
R	+5	+4	+1	-7	+1
MIX	-13	-6	-11	+8	+4

F_1 measure, back-off

lm \text	A	B	C	R	D
A	0.928	0.628	0.663	0.642	0.618
B	0.800	0.877	0.655	0.498	0.589
C	0.872	0.648	0.932	0.649	0.670
R	0.754	0.517	0.571	0.806	0.531
MIX	0.908	0.821	0.886	0.635	0.678

improvement of interpolated model

lm \text	A	B	C	R	D
A	+0.003	+0.001	+0.019	-0.007	+0.017
B	+0.005	+0.001	-0.002	+0.021	+0.002
C	+0.004	+0.011	+0.002	+0.006	+0.004
R	+0.020	+0.013	+0.014	+0.009	+0.014
MIX	+0.017	+0.054	+0.039	+0.001	+0.018

Precision of back-off model

lm \text	A	B	C	R	D
A	0.951	0.684	0.595	0.671	0.597
B	0.839	0.915	0.611	0.462	0.541
C	0.925	0.727	0.960	0.702	0.766
R	0.913	0.619	0.607	0.845	0.658
MIX	0.936	0.895	0.911	0.672	0.756

improvement of interpolated model

lm \text	A	B	C	R	D
A	+0.005	+0.032	+0.03	+0.015	+0.036
B	+0.008	+0.003	+0.003	+0.041	+0.006
C	+0.003	+0.012	-0.002	+0.023	-0.004
R	+0.019	+0.068	+0.052	+0.004	+0.049
MIX	+0.018	+0.026	+0.033	+0.006	+0.008

Recall of back-off model

lm \text	A	B	C	R	D
A	0.907	0.580	0.749	0.616	0.640
B	0.764	0.841	0.705	0.539	0.647
C	0.824	0.585	0.907	0.603	0.595
R	0.642	0.444	0.540	0.770	0.445
MIX	0.880	0.758	0.861	0.602	0.615

improvement of interpolated model

lm \text	A	B	C	R	D
A	+0.001	-0.019	+0.001	-0.026	-0.003
B	+0.002	0.000	-0.007	-0.003	-0.004
C	+0.005	+0.01	+0.003	-0.005	+0.009
R	+0.020	-0.013	-0.014	+0.015	-0.002
MIX	+0.017	+0.076	+0.045	-0.003	+0.025

Figure 9: Comparison of perplexity, F-measure, precision and recall of language models tested on other text types. The language models are on the rows and the text types on the columns. On the tables on the right is the difference in score achieved by the interpolated version of the model compared to the back-off version on the left.

OOV/ total of tokens in testing corpora					
lm \text	A	B	C	R	D
A	0.0028	0.0061	0.0123	0.0176	0.0130
B	0.0338	0.0006	0.0217	0.0230	0.0128
C	0.0092	0.0048	0.0026	0.0108	0.0096
R	0.0489	0.0215	0.0515	0.0080	0.0576
MIX	0.0023	0.0004	0.0017	0.0072	0.0072

Figure 10: Out-Of-Vocabulary words encountered during segmentation, relative to the total tokens of the testing set

the vocabulary.

A low perplexity also indicates better performance on the segmenting task to some extent. However, perplexities couldn't be used to predict the precision or recall. For example, the MIX-model has a very low perplexity of 13 on the B texts, and a significantly worse perplexity of 109 on the A text. However, both precision and recall of the MIX model on the A texts was better than on the B texts.

The perplexity measures how the language model models the whole text. In the segmentation task, however, the most important parts are how the sentence boundaries are modeled. In light of this, it is reasonable not to use perplexity to predict the performance on the segmentation task.

All of the models perform significantly worse on the test sets of other types of data. The best F-score of 0.932 (0.934 on interpolated) is achieved by the C language model on its own testing set. The best F-score of 0.872 (0.876 on interpolated) on another text type is achieved also by the C model on the A testset. Types with a small vocabulary, R and B, perform both poorly on the segmentation task and come across relatively many out-of-vocabulary words. The B model has a significantly low perplexity on its own testing set, which again suggest some repetition in the data.

Since the D-files were segmented automatically, it is important to consider the performance of the language model on the R set. The R set also provides an available reference point for the reader, since the other document types are completely unavailable. Let's recall that we deemed the R set to be closest to A and C sets in vocabulary; and the D set to be closest to C and B. In light of the segmentation results in Figure 9 The best models seem to be the A-, C- and MIX-models. Out of these the best model seems to be the one trained with C data.

The MIX-model that was trained on a combination of A,B and C performs slightly better in regards to perplexity. It also performs slightly better on the F-measure of D-files, and beats the C model in regards to recall on D. However, the C-model performs slightly better on the R-files in the segmentation task. It's also worth noting that even though the MIX-model has the advantage of having three times more training data than the other models, its performance is not significantly better than the C-model or the A-model in segmenting the R and D files. Taking this into account, it makes more sense to investigate another model further.

The A-model is the second best option to C. However, it performs slightly worse on the R and D-files in regards to F-measure. The C has especially better precision. The C model has also slightly less relative out-of-vocabulary words compared to A-model. This likely affected the performance as well.

We will use C data to do some further testing on the parameters of the language model. The C dataset performs well on both D and R-documents. It is also easier to control the training set, when using only one text type in training.

6.4 Comparison of back-off and interpolated model

The tests in previous subsection were repeated with a interpolated model. The tables in Figure 9 have on the right the difference of the scores given by the interpolated model, compared to the results of the back-off model.

The perplexity of each interpolated language model on its own training set improves by 5-9 units from the corresponding back-off model. This is in agreement with the claims that interpolated models usually perform slightly better in regards to perplexity. However, the perplexity does not improve on all other text types. Only the A and C models improve, or stay the same in regards to perplexity on all other text types. The MIX model improves its perplexity on A, B and C test sets, which are the same text types it was trained on.

Even though the interpolated model does not improve the perplexity of all models on all text types, it improves the segmentation performance of almost all of the models. Especially the interpolated models achieve a slightly better precision on almost all types.

An improvement of 0.001 in precision means that 0.1 percent more of the assigned sentence boundaries were correct. This is dependent on the number of sentence boundaries assigned by the model. A improvement of 0.001 in recall means that

0.1 percent more of the true sentence boundaries were found. Especially the MIX model benefited from interpolation by all measures. The increase of 0.017-0.076 of recall means the interpolated MIX model found 1.7-7.6% more of the true sentence boundaries compared to the back-off model.

6.5 Effect of retaining symbols and tokens

Now that we have defined the best text type to train the model on, we consider some other variables that affect the language model. The following tests are done on smaller testing sets and the exact results aren't as reliable. However, even with this small testing set we should be able to tell if there is a significant difference in performance between the methods.

One question in training the language models is how will the tokenization of the text affect the language model. Will keeping more punctuation always improve the performance or will keeping too much symbols create clutter and disturb the performance? To answer this question we test the performance of C language model on a small set of of 500 documents from C, D and R testing sets. The model is order 4, smoothed with interpolated Witten-Bell and trained on 90 000 C training files with various punctuation schemes. We also test a language model trained on the R training set on the full R testing set. The following schemes are experimented on:

0) Only lower-case words and no additional symbols or capitalization tags. Only tags used are the digit tags <d>.

1) Only words and the capitalization and digit tags. The tags are presented in Chapter 5.2.

2) 1 and inter-sentence punctuation tokens , : ; "

3) 1 and sentence-ending punctuation tokens . ? !

4) Both 2 and 3

5) Almost all possible symbols

6) All symbols, but no capitalization tags

All symbols and words are separated into their own tokens with a space. The results of the test are displayed in Figure 11.

We will first consider the language models of type R and C that were tested on their own testing set. The word-only approach achieves a F-score of 0.70-0.76. This

R-lm on R
(25,015 sentence boundaries)

symbols	P	R	F
0) word-only	0.7208	0.6897	0.7049
1)	0.8197	0.8187	0.8192
2) ,:;"	0.8414	0.8288	0.8351
3) .?!	0.8556	0.9134	0.8836
4) .?!,:;"	0.8483	0.8999	0.8733
5) .?!,:; "()#/-%*$&+=	0.8493	0.9024	0.8751
6) no <c>	0.8385	0.9018	0.8690

C-lm on C 500 files
(10,124 sentence boundaries)

symbols	P	R	F
0) word-only	0.7832	0.7484	0.7654
1)	0.8877	0.8731	0.8803
2) ,:;"	0.9073	0.8864	0.8967
3) .?!	0.9478	0.9346	0.9411
4) .?!,:;"	0.9420	0.9356	0.9388
5) .?!,:; "()#/-%*$&+=	0.9460	0.9357	0.9409
6) no <c>	0.9478	0.9278	0.9377

C-lm on D 500 files
(155,929 sentence boundaries)

symbols	P	R	F
0) word-only	0.5107	0.3175	0.3916
1)	0.6861	0.4820	0.5662
2) ,:;"	0.7211	0.5191	0.6037
3) .?!	0.7872	0.5605	0.6548
4) .?!,:;"	0.7978	0.5759	0.6689
5) .?!,:; "()#/-%*$&+=	0.7562	0.6005	0.6694
6) no <c>	0.7904	0.5528	0.6506

C-lm on R 500 files
(3,323 sentence boundaries)

symbols	P	R	F
0) word-only	0.4470	0.3978	0.4210
1)	0.6856	0.6713	0.6784
2) ,:;"	0.6975	0.6746	0.6859
3) .?!	0.7266	0.6662	0.6951
4) .?!,:;"	0.7292	0.6987	0.7136
5) .?!,:; "()#/-%*$&+=	0.7251	0.6954	0.7099
6) no <c>	0.7159	0.6120	0.6599

Figure 11: Comparison of language models trained on different punctuation schemes

approach corresponds to segmenting transcripts from a speech recognition system without any prosody clues, except that the document structure differs from what a spoken record would likely contain. The F-scores are large compared to the results from experiments in literature. This suggests that the data is significantly more uniform than the ones used in the experiments in literature. It is also worthwhile to note that the 0-method has longer history, since there are no additional capitalization tags that would take up the places of words.

Already the 1st method, in which there are only words and tokens for capitalization and digits, achieves a decent F-score of 0.82-0.88. We also note that the capitalization tokens improve the performance from the word-only approach: both in comparing 0 to 1 and 6 to 5.

For both the R and C language models the best F-score was achieved with the 3rd punctuation in which only the end-of-sentence punctuation marks are retained. The second best option was the 5th option where a almost all symbols were retained. The 5th option is the one used in the previous experiments.

When testing the C language model on the unseen D and R datasets, the best performing methods are the 4th and 5th. The difference in F-score between the best and worst methods are on the same type of data 0.0644 and 0.0608; and on the unseen types 0.1032 and 0.0352. This suggests choosing a good set of symbols to retain improves the performance, but already the words and capitalization and digit tokens provide information for the model.

6.6 Effect of training set size

The C model in previous sections was trained on a training set of 90 000 documents. It is important to consider whether using more training data would improve the performance of the model even further. A large training set is not a problem, since training the n-gram model is rather fast. However, it is also worth considering how many correctly segmented files would be enough to train a language model when we have limited resources.

In this section we will test how the number of training files affects the performance of the C language model. The language model is order 4, smoothed with interpolated Witten-Bell and the data is tokenized according to the 4th cleaning method presented in the previous Section 6.5. The model is tested on a small set of 500 C and D documents, and the testing set of R. The results of the experiment are

C-lm on C 500 files
(10,124 sentence boundaries in testing)

documents	sentences	P	R	F
1 000	19,684	0.9119	0.8508	0.8803
5 000	98,221	0.9336	0.8926	0.9126
10 000	194,608	0.9337	0.9083	0.9208
20 000	394,628	0.9376	0.9195	0.9285
50 000	981,907	0.9407	0.9306	0.9356
90 000	1,858,279	0.9420	0.9356	0.9388

C-lm on D 500 files
(155,929 sentence boundaries)

training	P	R	F
1 000	0.7960	0.4969	0.6119
5 000	0.8215	0.5325	0.6462
10 000	0.8163	0.5486	0.6562
20 000	0.8219	0.5513	0.6599
50 000	0.7843	0.5743	0.6631
90 000	0.7562	0.6005	0.6694

C-lm on R 500 files
(3,323 sentence boundaries)

training	P	R	F
1 000	0.7204	0.6190	0.6659
5 000	0.7392	0.6542	0.6941
10 000	0.7347	0.6761	0.7042
20 000	0.7320	0.6939	0.7124
50 000	0.7279	0.6948	0.7110
90 000	0.7292	0.6987	0.7136

Figure 12: Comparison of language models trained on different training set sizes

displayed in Figure 12.

For the C language model, tested on its own testing set, both precision and recall improved the more training data was used. The growth in performance is slow after 20 000 files: the improvement in F-score is 0.0482 from 1000 files to 20 000; and 0.0103 from 20 000 to 90 000. It would seem that more training data would not significantly improve the performance.

The C language model, tested on the D and R testing sets, improves its F-score the more training data is used. However, the improvement is slow after 20 000 training files: the improvements in F-score from 1000 to 20 000 are 0.048 and 0.0465; and the improvement from 20 000 files to 90 000 are only 0.0095 and 0.0012.

Especially the recall benefits from a larger training set. On the other hand, the precision even declines after 20 000 training files. This suggests that the language model assigns more sentence boundaries, the more training data was used.

6.7 Conclusions

In this chapter we experimented with the n-gram models from several points of view. We considered some experiments done with n-grams in literature and what are the best results humans achieve; how the order and smoothing methods affect the model; how the text type affects the language model's performance on other types of text; how the performance of a back-off model and interpolated model differ; how retaining different sets of punctuation and other symbols affect the performance; and finally how increasing the size of the training set improves the performance.

In the experiments by Stevenson and Gaizauskas [SG00] the human performance on the segmentation task without punctuation achieved an F-score of 0.96-0.97. The results achieved by the n-gram models even on punctuated corpora are, as expected, worse than this. The best F-score of 0.934 was achieved by the interpolated C model on the C testing set in Section 6.3. Changing the retained symbols was noted to slightly improve the performance in Section 6.5. However, the experiments on punctuation were tested on a smaller test set, which makes the exact F-score less reliable.

In Section 6.2 we experimented on the order and smoothing of the model. We decided to use the Witten-Bell smoothing method and an n-gram model of order 4. Higher order models achieved slightly better results, but the improvement in performance was small after the 4th order, whereas the model size grows significantly with each increase of order.

The experiments in Section 6.3 compared the performance of language models trained on one text type on another text type. Generally the performance was significantly worse on other text type's testing sets compared to the performance on the same type of testing data. This is the result of both the structural differences in what is considered to be a sentence and differences in vocabularies of the text types. It was noted that neither the comparison of vocabularies nor the perplexity could be used straightforwardly to predict the performance on the segmentation task, even though similarities in vocabulary and lower perplexity suggested better performance. In Section 6.4 the results were compared to repeating the experiment with an interpolated model. It was noted that the interpolated version performed better in most cases. The capitalization tags also improved the performance compared to the performance of the word-only approach, even when all punctuation was present.

In Chapter 6.5 we noted that using the word-only approach with no punctuation, the F-score was 0.70-0.77 on same type of data, and 0.39-0.42 on unseen data types of D and R. This was improved by adding the capitalization tokens, resulting in F-score of 0.82-0.88 on a language model tested on same type of data, and 0.57-0.68 on a unseen test data types. In most cases retaining as much as possible of the punctuation was better or almost as good as the other schemes.

Finally in Section 6.6 we tested how the training set size affects the performance of the model. It was noted that a larger amount of data improves the performance, but already a set of 20 000 C documents in training might yield almost as good results as a training set of 90 000 documents. The last two experiments were tested on a smaller testing set so the exact F-scores are not as reliable as the ones in Section 6.3.

Another aspect that would be beneficial to experiment on is restricting the words in the vocabulary. Currently the language model knows all the words in the training corpus and therefore doesn't learn any n-grams that contain a <unk> -unknown-word. When the model encounters words out of the vocabulary in the testing set it's forced to back of into the lower order n-grams. It might be very beneficial for the model to learn certain patterns that contain unusual words.

7 Other models for the sentence segmentation task

The hidden n-gram approach discussed in this thesis is a very simple and light-weight approach to the sentence segmentation task. Most systems use additional features such as part-of-speech tagging (POS) and more complicated models. POS tagging means assigning hidden word classes (noun, verb, adjective,...) to the words in a document. This is usually a preprocessing step in sentence segmentation and other NLP tasks. Automatic speech recognition (ASR) systems use also prosodic features such as pauses, pitch and tempo of speech.

Stolcke and Shriberg [SS96] noted in their experiment with n-gram models that adding part-of speech tags did not improve the performance from the word-only approach. They concluded that this might result from their part-of-speech tagger relying on the segmentation of the text. Both segmenting and tagging text might be too difficult for an n-gram model. Lafferty et al. [LMP$^+$01] experiment with a conditional random field (CRF) model that segments and labels sequence data with better results.

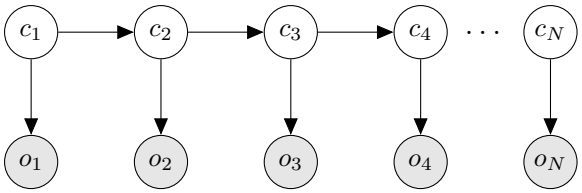

Figure 13: First order HMM. An arrow $X_i \rightarrow X_j$ denotes that the variable X_j is dependent on X_i

Since there is a continuous stream of documents to the AlphaSense system, many of the state of the art systems might be too slow to process the documents. Despite this, in this chapter we will present some alternative models used for the sentence segmentation task.

7.1 Hidden Markov model (HMM)

The Hidden Markov model is a Markov chain in which the sequence generated by the chain is hidden. Instead each state emits an observation.

In Figure 13 we can see the graphical model of a HMM. It is important to note that in a HMM, the observation o_i is independent from all other observations, when conditioned on the hidden state c_i. When modeling language this modeling approach is dubious, since it is not reasonable to assume words in a sequence to be independent from each other given the hidden classes such as word class or boundary information.

Shriberg et al. [SSHTT00] use an order 4 HMM in segmenting speech into sentences and topics. They assume the hidden states c_i are the sentence boundary or topic classes, and that the observations are words and prosodic features. They find the hidden sequence by finding the sequence c_1^N that

$$c_1^N = \arg\max_{c_1^N} p(c_1^N | o_1^N).$$

Their HMM relies on the hidden n-gram model presented in [SS96], so that for a bi-gram model, the probability that hidden state <B> emits the word v_i is p(<B>$|v_i$).

7.2 Conditional Random Fields (CRF)

Conditional Random field is an undirected graphical model. A undirected model assumes that the inference between connected variables goes both ways. This makes

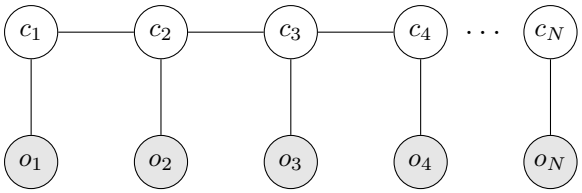

Figure 14: First order linear chain CRF. A line between X_i and X_j denotes that the variables are dependent on each other

the model more complex, as it is harder to make variables independent from other variables. Because of this, calculating the maximum likelihood parameters is not as straightforward as for directed graphical models such as Markov Chains or HMM. The CRF model's maximum likelihood parameters can be estimated using numerical methods such as gradient descent.

More precise theory on CRFs structure and training can be found in [Mur12] chapter 19.6 or [SM12].

A linear chain conditional random field is presented in Figure 14. The model is very similiar to the HMM, except the inference between variables goes both ways. The strength of a linear chain CRF compared to a directed model, is that the two-way inference allows events later in the chain to affect events before them. This allows for richer modeling of a sequence. In a Markov Chain or HMM, the events in the chain can't be affected by events later in the chain. However, the one way inference allows us to define fast dynamic algorithm, as seen in Chapter 4.3. For a undirected model, training and decoding a hidden path is significantly slower.

Conditional random fields are used in NLP problems such as POS tagging, topic segmentation, information extraction, named entity recognition and noun phrase chunking.

Liu et al. compare HMM and Maxent modeling approaches to the CRF approach for sentence boundary recognition in speech [LSSH05]. They conclude that the CRF models outperforms the HMM and Maxent models. They also note that training a CRF model is slow compared to the other models, especially when there are several features used to describe each event. Ho et al. [HCC$^+$16] address the problem of selecting features to use in a linear chain CRF model, to improve sentence boundary detection while maintaining the F-score. Ueffing et al. [UBV13] use a CRF for automatic punctuation recovery. Lafferty et al. [LMP$^+$01] experiment with a CRF that segments and labels sequence data.

8 Conclusions

In this thesis we took a look at language modeling and the problem of sentence segmentation. The thesis includes both theory on language modeling, as well as empirical results using n-gram language models. The first part of the thesis focused on the theory of language modeling and evaluating the results. In Chapter 2 we explained the probabilistic formulation of language models and the sentence segmentation task in mathematical terms. To evaluate the results, of both the segmentation task and language models in general, we presented the evaluation metrics of precision, recall, F-score and perplexity in Chapter 3. The n-gram model, training, smoothing and assigning sentence boundaries were discussed in detail in Chapter 4.

The second part of this thesis included empirical experiments on datasets provided by AlphaSense. The datasets used in the experiments were presented and compared in Chapter 5. It is worthwhile to note that a big challenge in language modeling and machine learning is working with real life corpora that might contain ambiguities and unnecessary or misleading parts. The performance of the n-gram models was tested in Chapter 6.

In the experiments we noted that using a 4-gram instead of a 2- or 3-gram improves the segmentation results. Increasing the model order from this does not seem to improve the results and increases the model size. We noted that the models can segment the same data type they were trained on, up to a F-score of 0.934. The models trained on different types of AlphaSense data segmented unseen AlphaSense documents significantly worse, with a F-score ranging from 0.591 to 0.872. The interpolated models were noted to perform slightly better than the back-off model. We also noted that increasing the training set size improves the performance of an n-gram model, but after 20 000 C documents in training, the improvement was slow.

The results using a language model trained on another type of data on the testing set of another type suggest that the n-gram model is not alone powerful enough to do the segmentation on the unseen datatype D. Further, the experiments on punctuation and training set size suggest that the results can't be significantly improved by using a different tokenization scheme or bigger training set size.

In the final Chapter 7 we presented some alternative models used for sentence segmentation and language modeling. Some specifics and examples of the SRILM toolkit were included in the Appendixes of the thesis.

References

BBL98 Beeferman, D., Berger, A. and Lafferty, J., Cyberpunc: A lightweight punctuation annotation system for speech. *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2. IEEE, 1998, pages 689–692.

CG96 Chen, S. F. and Goodman, J., An empirical study of smoothing techniques for language modeling. *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1996, pages 310–318.

CT06 Cover, T. M. and Thomas, J. A., *Elements of Information Theory, 2nd edition*. Wiley-interscience, 2006.

GoS00 Gotoh, Y. and Renals, S., Sentence boundary detection in broadcast speech transcripts. *Proceedings of the International Speech Communication Association*, 2000.

HCC+16 Ho, T.-N., Chong, T. Y., Chng, E. S. et al., Improving efficiency of sentence boundary detection by feature selection. *Asian Conference on Intelligent Information and Database Systems*. Springer, 2016, pages 594–603.

HM15 Hirschberg, J. and Manning, C. D., Advances in natural language processing. *Science*, 349,6245(2015), pages 261–266. URL `http://science.sciencemag.org/content/349/6245/261`.

Kat87 Katz, S., Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35,3(1987), pages 400–401.

LMP+01 Lafferty, J., McCallum, A., Pereira, F. et al., Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the Eighteenth International Conference on Machine Learning, ICML*, 2001, pages 282–289.

LSSH05 Liu, Y., Stolcke, A., Shriberg, E. and Harper, M., Using conditional random fields for sentence boundary detection in speech. *Proceedings of*

the 43rd Annual Meeting on Association for Computational Linguistics, ACL '05, 2005, pages 451–458.

MS99 Manning, C. D. and Schütze, H., *Foundations of statistical natural language processing*. MIT Press, 1999.

Mur12 Murphy, K. P., *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

NC11 Nadkarni, Prakash M, L. O.-M. and Chapman., W. W., Natural language processing: An introduction. *Journal of the American Medical Informatics Association*. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3168328/pdf/amiajnl-2011-000464.pdf`.

S+02 Stolcke, A. et al., Srilm-an extensible language modeling toolkit. *Interspeech*, 2002.

SG00 Stevenson, M. and Gaizauskas, R., Experiments on sentence boundary detection. *Proceedings of the sixth conference on Applied natural language processing*. Association for Computational Linguistics, 2000, pages 84–89.

SM12 Sutton, C. and McCallum, A., An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4,4(2012), pages 267–373.

SS96 Stolcke, A. and Shriberg, E., Automatic linguistic segmentation of conversational speech. *Fourth International Conference on Spoken Language ICSLP 96. Proceedings.*, volume 2. IEEE, 1996, pages 1005–1008.

SSHTT00 Shriberg, E., Stolcke, A., Hakkani-Tür, D. and Tür, G., Prosody-based automatic segmentation of speech into sentences and topics. *Speech communication*, 32,1(2000), pages 127–154.

SZWA11 Stolcke, A., Zheng, J., Wang, W. and Abrash, V., Srilm at sixteen: Update and outlook. *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*, volume 5, 2011.

UBV13 Ueffing, N., Bisani, M. and Vozila, P., Improved models for automatic punctuation prediction for spoken and written text. *INTERSPEECH*, 2013, pages 3097–3101.

WB91 Witten, I. H. and Bell, T. C., The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37,4(1991), pages 1085–1094.

A The SRILM toolkit

SRILM[5] is a toolkit for natural language processing, developed at Speech Technology and Research Laboratory SRI International. The SRILM toolkit was first released in 1999 and has been updated since.

The SRILM is a collection of C++ libraries and executable programs, including functions for training and applying n-gram language models. The SRILM toolkit is open source and available for licensing. For an overview of SRILM functionalities, refer to the articles [S+02] [SZWA11] by Stolcke et al.

The toolkit has an active user mailing list srilm-user@speech.sri.com

[5]Download and installation instructions at `http://www.speech.sri.com/projects/srilm/`

B The ARPA format

The SRILM language models are saved in a standard backoff n-gram format called the ARPA format. The probabilities are stored in $\log_{10}$. The logarithim allows us to sum the probabilities together instead of multiplication. It also makes small decimals larger and therefore easier to store. The back-off model is explained in Chapter 4.2.

Denote the number of 1-grams as N1, and number of 2-grams in the model as N2, and so forth. The back-off weight of n-gram w1..wn is denoted as bow(w1..wn). The ARPA format has the following form:

```
\data\
ngram 1=N1
ngram 2=N2
...
ngram n=Nn

\1-grams:
log(p(w))          w                    bow()
...

\2-grams:
log(p(w2 | w1))  w1 w2                  bow()
...

\n-grams:
log(p(w_n|w1...w_{n-1}))                w1 ... wn
...

\end\
```

C Creating a language model in SRILM

As an example corpus we use an text extract from Alice's Adventures in Wonderland by Lewis Carroll [6]

```
1 "The Queen of Hearts, she made some tarts,
2 All on a summer day;
3 The Knave of Hearts, he stole those tarts
4 And took them quite away!"
```

First we tokenize the text according to the scheme presented in Chapter 5.2 and save it into file clean_alice.txt.

```
1 <B> " <c> the <c> queen of <c> hearts , she made some tarts
    , <B> <c> all on a summer day ; <B> <c> the <c> knave of
    <c> hearts , he stole those tarts <B> <c> and took them
    quite away ! " <B>
```

The SRILM toolkit can be used to create a maximum likelihood language model order 2 based on the corpora with the following command:

```
$ ngram-count -text clean_alice.txt -order 2 \
> -lm alice.lm -addsmooth 0 -no-sos -no-eos
```

The language model in ARPA format is saved into file alice.lm. The -order parameter defines the order of the model. For orders above 3 it is recommended to use the make-big-lm wrapper function instead[7].

The -addsmooth 0 parameter tells the function to use additive smoothing with adding 0 to each count. This corresponds to the maximum likelihood model. The back-off version of Witten-Bell smoothing can be used by replacing the -addsmooth 0 with -wbdiscount and the interpolated version by adding -interpolated. More smoothing methods can be found in the SRILM toolkit's documentation ngram-discount[8].

The -no-sos -no-eos parameters prevent the function from adding the <s> and </s> tokens into the beginning and end of the lines in the training corpus. Despite this, the beginning and end of sentence tokens <s> </s> are included in the vocabulary

[6] `https://www.gutenberg.org/ebooks/19033`

[7] http://www.speech.sri.com/projects/srilm/manpages/training-scripts.1.html

[8] http://www.speech.sri.com/projects/srilm/manpages/ngram-discount.7.html

automatically.

Below we can see the bigram models in ARPA format. The bigram model on the left is the maximum likelihood model, and the one on the right is the Witten-bell back-off discounted model. The value -99 is a dummy value that denotes negative infinity (log(0)), the zero probability.

maximum likelihood model

```
\data\
ngram 1=30
ngram 2=38

\1-grams:
-1.662758       !       -99
-1.361728       "       -99
-1.185637       ,       -99
-1.662758       ;       -99
-99     </s>
-0.9637879      <B>     -99
-0.7596679      <c>     -99
-99     <s>
-1.662758       a       -99
-1.662758       all     -99
-1.662758       and     -99
-1.662758       away    -99
-1.662758       day     -99
-1.662758       he      -99
-1.361728       hearts  -99
-1.662758       knave   -99
-1.662758       made    -99
-1.361728       of      -99
-1.662758       on      -99
-1.662758       queen   -99
-1.662758       quite   -99
-1.662758       she     -99
-1.662758       some    -99
-1.662758       stole   -99
-1.662758       summer  -99
-1.361728       tarts   -99
-1.361728       the     -99
-1.662758       them    -99
-1.662758       those   -99
-1.662758       took    -99
```

Witten-Bell discounted model

```
\data\
ngram 1=30
ngram 2=38

\1-grams:
-1.869232       !       -0.2891307
-1.568202       "       -0.2171281
-1.39211        ,       -0.2578731
-1.869232       ;       -0.2706473
-0.4220737      </s>
-1.170262       <B>     -0.4140695
-0.9661418      <c>     -0.318289
-99     <s>
-1.869232       a       -0.2951211
-1.869232       all     -0.2951211
-1.869232       and     -0.2951211
-1.869232       away    -0.2951211
-1.869232       day     -0.2951211
-1.869232       he      -0.2951211
-1.568202       hearts  -0.4591479
-1.869232       knave   -0.2891307
-1.869232       made    -0.2951211
-1.568202       of      -0.4274335
-1.869232       on      -0.2951211
-1.869232       queen   -0.2891307
-1.869232       quite   -0.2951211
-1.869232       she     -0.2951211
-1.869232       some    -0.2891307
-1.869232       stole   -0.2951211
-1.869232       summer  -0.2951211
-1.568202       tarts   -0.2513422
-1.568202       the     -0.4274335
-1.869232       them    -0.2951211
-1.869232       those   -0.2891307
-1.869232       took    -0.2951211
```

```
\2-grams:
0       ! "
-0.30103        " <B>
-0.30103        " <c>
-0.4771213      , <B>
-0.4771213      , he
-0.4771213      , she
0       ; <B>
-0.60206        <B> "
-0.1249387      <B> <c>
-0.90309        <c> all
-0.90309        <c> and
-0.60206        <c> hearts
-0.90309        <c> knave
-0.90309        <c> queen
-0.60206        <c> the
0       a summer
0       all on
0       and took
0       away !
0       day ;
0       he stole
0       hearts ,
0       knave of
0       made some
0       of <c>
0       on a
0       queen of
0       quite away
0       she made
0       some tarts
0       stole those
0       summer day
-0.30103        tarts ,
-0.30103        tarts <B>
0       the <c>
0       them quite
0       those tarts
0       took them

\end\
```

```
\2-grams:
-0.30103        ! "
-0.60206        " <B>
-0.60206        " <c>
-0.7781513      , <B>
-0.7781513      , he
-0.7781513      , she
-0.30103        ; <B>
-0.7781513      <B> "
-0.30103        <B> <c>
-1.146128       <c> all
-1.146128       <c> and
-0.845098       <c> hearts
-1.146128       <c> knave
-1.146128       <c> queen
-0.845098       <c> the
-0.30103        a summer
-0.30103        all on
-0.30103        and took
-0.30103        away !
-0.30103        day ;
-0.30103        he stole
-0.1760913      hearts ,
-0.30103        knave of
-0.30103        made some
-0.1760913      of <c>
-0.30103        on a
-0.30103        queen of
-0.30103        quite away
-0.30103        she made
-0.30103        some tarts
-0.30103        stole those
-0.30103        summer day
-0.60206        tarts ,
-0.60206        tarts <B>
-0.1760913      the <c>
-0.30103        them quite
-0.30103        those tarts
-0.30103        took them

\end\
```

D Example of calculating the probability using the back-off model

In this Appendix we will show how to use the back-off model to calculate the probability of a sentence. We use the 2-gram witten-bell discounted language model presented in the previous appendix. The sentence we use as an example is "stole those hearts". The probabilities are in stored as log_10 of the probability. For simplicity we will refer to the log probability as probability.

1) The (log) probability of word "stole" is -1.869. This is written on the 1-gram section of the ARPA model.

2) The probability of transitioning from word "stole" to word "those" is -0.301 This is written on the 2-gram section on the left of "stole those"

3) The transition from "those" to "hearts" was not observed during training. To calculate the probability, we need to use basking-off. The back-off weight of "those" is -0.289. This is in the 1-gram section on the right of the word. And the probability of word "hearts" is -1.568. The back-off probability is now $-1.568 - 0.289 = -1.857$

The total probability of the sentence is $-1.869 - 0.301 - 1.857 = -4.027$. In the non-logarithmic scale this is $10^{-4.027} \approx 0.00009397$.

E Perplexity in SRILM

SRILM ignores the out of vocabulary words in calculating perplexity. The equation is uses is

```
ppl = 10^(-logprob / (words - OOVs + sentences))
ppl1 = 10^(-logprob / (words - OOVs))
```

where OOV denotes number of out-of-vocabulary words.

We can calculate the perplexity of a document, using the maximum likelihood language model from Appendix C, using the function ngram and defining variables -lm for the language model in ARPA format, and -ppl for the document

```
$ ngram -ppl testing.txt -lm alice.lm -debug 2 -no-sos -no-eos
stole those hearts
        p( stole |  )    = [1gram] 0.02173912 [ -1.662758 ]
        p( those | stole ...)    = [2gram] 1 [ 0 ]
        p( hearts | those ...)   = [1gram] 0 [ -inf ]
0 sentences, 3 words, 0 OOVs
1 zeroprobs, logprob= -1.662758 ppl= 6.782331 ppl1= 6.782331
```

The sequence "those hearts" has not occurred in the training, so its probability is zero. SRILM ignores the zero-probability, but in truth the perplexity is infinite.

F Example of segmentation using Viterbi

The theory of the Viterbi algorithm was presented in Chapter 4.3. We will now present an example of the segmentation using the ML model from previous Appendix and a new piece of text.

We define the hidden vocabulary to be the boundary token <B>, and a no-event. The sequence to be segmented with the unsmoothed 2-gram model from Appendix B is "tarts , <c> knave".

```
$ cat segment_this.txt
 tarts , <c> knave
$ cat hidden_vocab.txt
<B>
$ hidden-ngram -lm alice.lm -text segment_this.txt \
> -hidden-vocab hidden_vocab.txt
tarts , <B> <c> knave
```

The algorithm assigns a sentence boundary between , and <c>. You can see the steps of the computation SRILM does by defining the option -debug 3 in the hidden-ngram function.

We choose the events as either <B> or no-event so that the probability
p(tarts e_1 , e_2 <c> e_3 knave)
is maximised. We already saw that the maximum likelihood sequence is
"tarts *no-event* , <B> <c> *no-event* knave".

	1 tarts	2 e_1 $\delta(1)$	3 ,	4 e_2 $\delta(2)$	5 <c>	6 e_3 $\delta(3)$	7 knave	8 e_4 $\delta(4)$
<B>		-0.3	-inf	-0.78	-0.90	-inf	-inf	-inf
no-event	0	0	-0.3	-0.3	-inf	-0.9	-1.8	**-1.8**

1) In the first index is just the word "tarts". The probability is initialized to be 1. In the logarithmic scale this is 0.

2) In index 2 is the first potential event. The event probability is
$\delta_{<B>}(1) = \log p(< B > |tarts) = -0.30$.
and the no-event case stays the same probability $\delta_{no-event}(1) = 0$.
An example of calculating the probabilities from the ARPA model was presented in

Appendix D.

3) In the third index is the word ",". The probability of the whole sequence so far is

$$\begin{cases} \log p(,|tarts < B >) & = \log p(< B > |tarts) + \log p(,| < B >) \\ & = -0.3 - inf \\ & = -inf, \text{ or} \\ \log p(,|tarts) & = -0.3 \end{cases}$$

4) At hidden event 2 we again have two choices: the event is either <B> or a no-event. For both events, we choose dynamically the path that maximizes the probability of the sequence so far

$$\begin{aligned} \delta_{<B>}(2) &= \max \begin{cases} \delta_{no-event}(1) + \log p(\text{<B> | tarts ,}) \\ \delta_{<B>}(1) + \log p(\text{<B> | <B> , }) \end{cases} \\ &= \max \begin{cases} 0 - 0.3 - 0.48 \\ -0.3 - inf \end{cases} \\ &= -0.78, \text{ from no-event} \\ \delta_{no-event}(2) &= \max \begin{cases} \delta_{no-event}(1) + \log p(, \text{ | tarts}) \\ \delta_{<B>}(1) + \log p(\text{ , | <B>}) \end{cases} \\ &= \max \begin{cases} 0 - 0.3 \\ -0.3 - inf \end{cases} \\ &= -0.3, \text{ from no-event} \end{aligned}$$

6) Similarly for the third event

$$\begin{aligned}
\delta_{<B>}(3) &= \max \begin{cases} \delta_{no-event}(2) + \log p(<\text{B}> \mid , <\text{c}>) \\ \delta_{<B>}(2) + \log p(<\text{B}> \mid <\text{c}>) \end{cases} \\
&= \max \begin{cases} -0.3 - inf \\ -0.78 - inf \end{cases} \\
&= -inf, \text{ from no-event} \\
\delta_{no-event}(3) &= \max \begin{cases} \delta_{no-event}(2) + \log p(<\text{c}> \mid ,) \\ \delta_{<B>}(2) + \log p(<\text{c}> \mid <\text{B}> <\text{c}>) \end{cases} \\
&= \max \begin{cases} -0.3 - inf \\ -0.78 - 0.12 \end{cases} \\
&= -0.90, \text{ from } <\text{B}>
\end{aligned}$$

8) Finally, the Viterbi maximum likelihood path, indicated in red, is traced back from the last state that has higher probability.

The computation is same for larger order n-grams and multiple hidden states. If the order is larger, the probability is calculated using the appropriate number of previous states, and the stored traceback is the (n-1)-gram that maximizes the probability, instead of just the previous state. If there are multiple hidden states, the algorithm calculates the probabilities for each of them appearing at event e. In the trellis this corresponds to having an additional row for each hidden state.